

Zum Entwurf und zur Anwendung programmierbarer Prozessoren für die digitale Signalverarbeitung

von
Hans-Jürgen Schloss

Erlangen, 1980

1979

1080 W 7

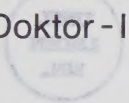


Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41546213_0007

Zum Entwurf und zur Anwendung programmierbarer Prozessoren für die digitale Signalverarbeitung

Der Technischen Fakultät
der Universität Erlangen-Nürnberg
zur Erlangung des Grades
Doktor-Ingenieur



vorgelegt von
Hans-Jürgen Schloss

Erlangen, 1980



Als Dissertation genehmigt von der Technischen Fakultät
der Universität Erlangen - Nürnberg

Tag der Einreichung: 19.09.1979

Tag der Promotion: 14.12.1979

Dekan: Prof.Dr.-Ing. H. Niemann

Berichterstatter: Prof.Dr.-Ing. W. Schüßler
Prof.Dr.-Ing. D. Seitzer

Herrn Prof.Dr.-Ing. H.W. Schüßler gebührt mein Dank für die interessante und aktuelle Aufgabenstellung. Er hat die Arbeiten in jeder Phase mit Rat und Tat unterstützt und mit wertvollen Hinweisen gefördert, er war somit wesentlich am Gelingen beteiligt.

Für die Übernahme des Zweitreferates und das damit bekundete Interesse, sowie für hilfreiche Diskussionen gilt mein besonderer Dank auch Herrn Prof.Dr.-Ing. D. Seitzer.

Schließlich sei auch der Deutschen Forschungsgemeinschaft gedankt, die durch die Bereitstellung finanzieller Mittel einen Teil der Arbeiten förderte.

I N H A L T S V E R Z E I C H N I S

	Seite
1. EINFÜHRUNG	1
2. ANWENDUNGSKLASSEN DIGITALER SYSTEME	3
2.1 Echtzeitsignalverarbeitung	3
2.2 Datennachverarbeitung	5
2.3 Signalprozessoren als Satellitenrechner	5
3. ZERLEGBARKEIT VON SIGNALVERARBEITUNGSLGORITHMEN	7
3.1 Signalverarbeitung in Teilprozessoren	7
3.1.1 Prinzip	7
3.1.2 Rechenzeitabschätzung	9
3.2 Algorithmenbeispiele	10
3.2.1 Nichtrekursive digitale Filter	10
3.2.1.1 Filtergleichung	10
3.2.1.2 Sonderfälle	11
a) Linearphasige Filter	11
b) Korrelationsfunktion	12
3.2.1.3 Darstellung im Z-Bereich	13
3.2.1.4 Realisierung der Filtergleichung mit Teilprozessoren	15
3.2.1.5 Zur Programmierung des nicht- rekursiven Filters	18
a) Schleifenfreie Programmierung	18
b) Indizierung	19
3.2.2 Rekursive Digitalfilter	23
3.2.2.1 Beispiel für die Struktur eines rekursiven Blockes 2. Grades	24
3.2.2.2 Probleme bei Festkommarealisierung	26
a) Grenzzyklen	28
b) Koeffizientennormierung	29
3.2.2.3 Teilprozessorstruktur	30
3.2.3 Die schnelle Fouriertransformation	32
3.2.3.1 Realisierung eines FFT-Butterflies in einem Teilprozessor	35
3.2.3.2 Berechnung der FFT mit Teil- prozessoren	39
a) Verwendung eines Teilprozessors	39

b) Verwendung von m Teilprozessoren	40
c) Verwendung von $M/2$ ldm Teilprozessoren	42
3.2.4 Potenzreihenentwicklung	43
3.2.5 Nichtrekursive zweidimensionale Filterung vom Blockgrad 0	45
3.2.5.1 Verarbeitungsprinzip	46
3.2.5.2 Modifikation des Algorithmus	49
a) Realisierung durch ein eindimensionales, invariantes nichtrekursives Filter	49
b) Realisierung bei Ausnutzung von Symmetrien	56
c) Realisierung mit variablen Filterkoeffizienten	58
d) Realisierung der Modifikationsmatrizen	64
3.2.5.3 Rechenaufwand	66
3.2.6 Rekursive Bildfolgenverarbeitung vom Blockgrad 1	69
3.2.6.1 Prinzip	69
3.2.6.2 Realisierung durch ein variables, eindimensionales, rückgekoppeltes nichtrekursives Filter	70
a) Struktur	71
b) Verwendung von Teilprozessoren	72
3.2.7 Zusammenfassung	74
3.2.7.1 Universeller Teilprozessor	75
3.2.7.2 Teilprozessor mit erhöhter Zahl von Rechenwerken	78
4. SPEZIELLE ANFORDERUNGEN AN SIGNALPROZESSOREN	82
4.1 Adressierung	82
4.1.1 Operandenadressierung	82
4.1.2 Indizierte Adressierung	83
4.1.3 Bitreversal	85

	Seite
4.2 Synchronisationsmechanismen	86
4.2.1 Taktsynchronisation	86
4.2.2 Unterbrechungssynchronisation	88
4.3 Datenverknüpfungen	89
5. MÖGLICHKEITEN FÜR DIE ARCHITEKTUR VON SIGNALPRO- ZESSOREN BEI VERWENDUNG NEUERER BAUELEMENTE	90
5.1 Pipelineprinzip	91
5.1.1 Problemstellung	91
5.1.2 Pipelinelösung	93
5.1.3 Theoretische Geschwindigkeitsgrenze	94
5.2 Zellrechnerstruktur	95
5.2.1 Teilprozessorstruktur	95
5.2.1.1 Minimalsystem	95
5.2.1.2 Erweitertes System	98
5.2.1.3 Realisierung	99
a) Einsatz von Pipelineregistern	99
b) Bauteileaufwand	101
5.2.2 Kaskadierung der Teilprozessoren	103
5.2.2.1 Teilprozessoren	103
5.2.2.2 Steuerwerke	103
5.2.2.3 Gemeinsamer Speicher	105
5.2.2.4 Steuerrechner	106
5.2.3 Anwendungsbeispiele	107
5.3 Universeller digitaler Signalprozessor	108
5.3.1 Systemarchitektur	109
5.3.1.1 Teilprozessorstruktur	111
a) Koeffizientenspeicher	114
b) Nichtlinearitäten	115
c) Steuerregister	116
d) Register für das Statuswort	117
e) Steuerung	118
5.3.1.2 Zentraleinheit	118
a) Aufgaben und Struktur	118
b) Befehlsbearbeitung	121
c) Realisierung der wichtigsten Funktionsblöcke	132

5.3.1.3 Speichermoduln	145
5.3.1.4 Ein-/Ausgabe	146
5.3.1.5 Konsolensteuerung	147
5.3.1.6 Technische Daten	148
a) Arithmetische Einheit	148
b) Zentraleinheit	148
c) Speicher	149
d) Ein-/Ausgabe	149
5.3.2 Anwendungsbeispiele	150
5.3.2.1 Eindimensionale Signalverarbeitung	150
5.3.2.2 Mehrdimensionale Signalverarbeitung	151
6. ZUSAMMENFASSUNG	154
7. LITERATURVERZEICHNIS	156

Liste der wichtigsten Formelzeichen

A	Beschreibungsmatrix für einen Teilprozessor
$\left. \begin{matrix} a \\ b \\ c \end{matrix} \right\}$	allgemeiner Koeffizient
E	Einheitsmatrix
f	Frequenz
G	Diagonalmatrix
$H(z)$	Übertragungsfunktion
$h_0(k)$	Impulsantwort
k	normierte Zeitvariable
ℓ_1, ℓ_2	Koordinaten zweidimensionaler Datenfelder
L_1, L_2	Feldausdehnungen zweidimensionaler Datenfelder
M	Modifikationsmatrix
n	Filtergrad, Laufparameter
$\mathbb{N}$	Menge der natürlichen Zahlen ohne Null
$\mathbb{N}_0$	Menge der natürlichen Zahlen mit Null
T	Abtastintervall, Verzögerungsintervall
T_R	Rechenzeit
T_M	Multiplikationszeit
T_A	Additionszeit
T_{TR}	Datentransportzeit
τ	Verzögerungszeit
$u(k), U(z)$	Eingangssignal, z-transformiertes Eingangssignal
$\left. \begin{matrix} v(k), V(z) \\ x(k), X(z) \end{matrix} \right\}$	Zwischensignal, z-transformiertes Zwischensignal
z	Binärstelle
$y(k), Y(z)$	Ausgangssignal, z-transformiertes Ausgangssignal
z	komplexe Frequenzvariable
<ADR>	bedeutet: Inhalt der Adresse ADR



1. Einführung

Das Thema der vorliegenden Arbeit ist der Entwurf und die Anwendung von digitalen Signalprozessoren. Unter einem digitalen Signalprozessor sei ein programmierbares Rechnersystem verstanden, dessen innere Struktur sich für die Ausführung der Algorithmen eignet, die durch die Theorie diskreter Systeme bereitgestellt werden [1]. Es handelt sich also primär um eine Untersuchung der Zusammenhänge zwischen den Algorithmen und den realisierenden Prozessorstrukturen. Durch eine so begründete Spezialisierung wird es möglich, mit kompakten Systemen sehr hohe Rechenleistungen zur Verfügung zu stellen, die sonst nur mit Großrechnern erreichbar sind.

Eine Reihe von Signalprozessorkonzepten entstand als industrielle Entwicklungen. Häufig werden sie als Peripherieprozessoren zu einem Allzweckrechner entworfen (z.B. [2, 3, 4, 5]), oder sie sind selbständige Einheiten mit vergleichsweise geringen Rechengeschwindigkeiten (z.B. [6, 7]). Auch aus dem Bereich der Forschung sind Konzepte bekannt geworden, die z.T. außerordentlich aufwendig (z.B. [8, 9, 10]) oder spezialisiert sind (z.B. [11]). Seit dem Entstehen dieser Systeme hat sich die Großintegrationstechnik stürmisch weiterentwickelt, so daß mittlerweile der Aufbau von schnellen, flexiblen und universell einsetzbaren digitalen Signalprozessoren bei nur geringem Hardwareaufwand möglich ist.

Hierzu ist in erster Linie die Entwicklung hochintegrierter Multiplizier- und Speicherbausteine zu nennen, sowie die Bereitstellung bipolarer Mikroprozessorelemente, die hohe Rechengeschwindigkeiten erlauben.

Es bleiben hier MOS-Mikroprozessoren mit festem Befehlssatz außer Betracht, da ihre Rechengeschwindigkeit im Vergleich zu den oben erwähnten Bausteinen gering ist und sich deshalb ihr Einsatz in der Echtzeitsignalverarbeitung häufig verbietet.

Das Ziel dieser Arbeit ist deshalb das Auffinden von Rechnerstrukturen, die den Möglichkeiten der neueren, schnell arbeitenden Logikelemente einerseits, und den wichtigsten Signalverarbeitungsalgorithmen andererseits angepaßt sind.

Dazu soll hier in drei Stufen vorgegangen werden:

- Nach einer Definition der Anwendungsklassen digitaler Signalprozessoren werden für eine Auswahl von Algorithmen Untersuchungen darüber angestellt, ob sie sich in Teilaufgaben zergliedern lassen, die mit einer möglichst einfachen Prozessorstruktur realisierbar sind.
- Danach werden einige Algorithmen- und Anwendungsspezifische Anforderungen an Signalprozessoren zusammengestellt, die durch die erste Untersuchung nicht erfaßt wurden.
- Schließlich werden zwei z.T. realisierte Strukturen für Signalprozessoren mit hochintegrierten Bausteinen angegeben und detailliert beschrieben. Hierbei wird auch die Anwendung der aufgebauten Systeme für konkrete Aufgaben gezeigt.

Zur Abgrenzung des Themas sei angemerkt, daß die hier untersuchten und vorgeschlagenen Strukturen für Rechenwerke ausschließlich unter dem Aspekt ihrer Anwendung in der digitalen Signalverarbeitung entwickelt werden. Es handelt sich also nicht um Realisierungen digitaler Universalrechner, wenngleich auch zur bequemen Handhabung der Systeme ein gewisses Maß an Universalität notwendig ist. Weiterhin bleiben festverdrahtete Systeme außer Betracht.

Zum Problemkreis "Programmentwicklung" sei bemerkt, daß hier ausschließlich die Programmierung auf Maschinenbefehls- bzw. Mikroprogrammebene betrachtet wird, da es kein Programmierverfahren gibt, das höhere Rechengeschwindigkeiten zuläßt. Das Erreichen möglichst kurzer Programmlaufzeiten ist eines der Hauptziele dieser Arbeit.

2. Anwendungsklassen digitaler Systeme

Beim Entwurf eines Signalverarbeitungssystems ist neben seiner Eignung für die einschlägigen Algorithmen darauf zu achten, daß der Datenverkehr nach außen den arithmetischen Fähigkeiten angepaßt ist. Man kann hierbei verschiedene Anwendungsklassen unterscheiden, die kurz dargestellt werden.

2.1 Echtzeitsignalverarbeitung

Das Blockschaltbild einer Echtzeitsignalverarbeitungsanlage ist in Bild (2.1) dargestellt.

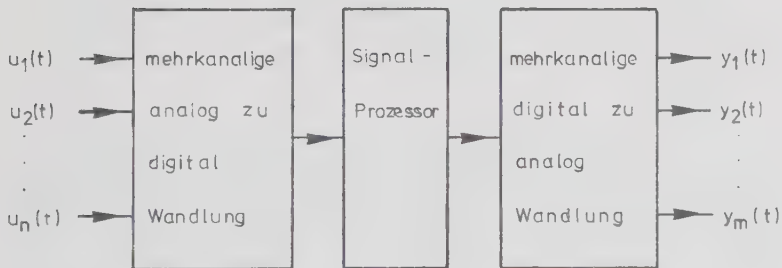


Bild (2.1) Allgemeines Blockschaltbild zur digitalen Echtzeitsignalverarbeitung

Es werden dabei die am System anliegenden Signalquellen

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) \\ u_2(t) \\ \vdots \\ u_n(t) \end{bmatrix}$$

umgeformt in einen Satz von Ausgangssignalen

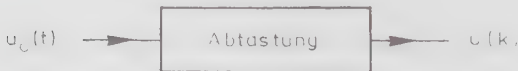
$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_m(t) \end{bmatrix} \quad ,$$

d.h. der Vektor $\mathbf{y}(t)$ ist die Systemreaktion auf den Eingangsvektor $\mathbf{u}(t)$:

$$\mathbf{y}(t) = \mathcal{S}\{\mathbf{u}(t)\} \quad .$$

Echtzeitsignalverarbeitung bedeutet, daß der Vektor $\mathbf{u}(t)$ der Eingangssignale sofort verarbeitet wird und der Ausgangsvektor $\mathbf{y}(t)$ so schnell errechnet wird, daß die im Signalprozessor vorhandenen Speicher durch einlaufende Signalabtastwerte nicht überlaufen können, daß also ein Datenstau vermieden wird. Dies ist äquivalent zu der Aussage, daß das System seine Aufgabe für eine beliebig lange Zeit erfüllen kann.

Eine digitale Verarbeitung analoger, zeitvariabler Größen erfordert deren Abtastung und Analog/Digitalwandlung. Dadurch entsteht aus einem kontinuierlichen Signal $u_0(t)$ eine Signalfolge $u(k)$ [13]:



Wobei gilt

$$u(k) = u_0(t = k \cdot T)$$

mit T = Abtastintervall

und k = ganzzahlig

2.2 Datennachverarbeitung

Bei der Anwendung eines Signalprozessors für die Nachverarbeitung von Signalabtastwerten (Off-Line-Verarbeitung) spielt die Rechengeschwindigkeit nicht die existentielle Rolle wie bei der Echtzeitverarbeitung. Der Signalprozessor konkurriert hierbei direkt mit dem Universalrechner und kann seine Existenzberechtigung durch Vorteile bei der Programmierung und in kürzeren Programmlaufzeiten gegenüber dem Universalrechner finden.

2.3 Der Signalprozessor als Satellitenrechner

Die dritte Anwendungsklasse eines Signalprozessors ist in der Unterstützung eines Universalrechners zu sehen (Bild 2.2).

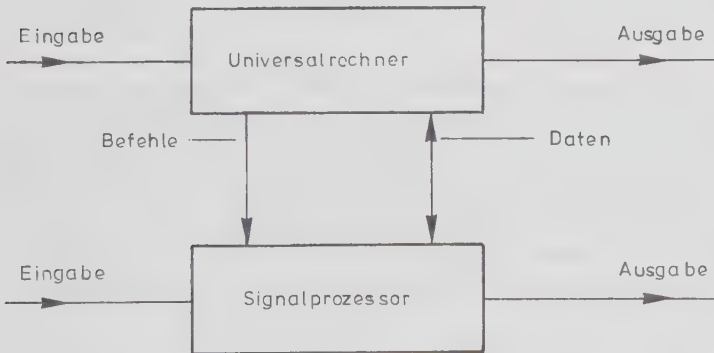


Bild (2.2) Typische Konfiguration eines Universalrechners mit einem externen Signalprozessor

Die Aufgabe des Signalprozessors besteht hier im wesentlichen in der Entlastung des Universalrechners von den meist sehr

rechenintensiven Algorithmen der Signalverarbeitung. Ein typisches Beispiel ist die Berechnung der schnellen Fouriertransformation in einem externen Prozessor, wobei sich die Rechenzeit z.B. um den Faktor 50 ... 100 verkürzen kann. Das ist allerdings nur dann zu erreichen, wenn die Zeit für den Datenaustausch zwischen dem Universalrechner und dem Signalprozessor klein bleibt gegenüber der Rechenzeit des Prozessors. Tatsächlich liegt hier ein wesentliches Problem in der Kopplung der Geräte.

In [12] wird angegeben, daß für den Transport eines Datenwortes aus dem Kernspeicher eines Rechners PDP 11/45 in einen externen Prozessor bei

- a) Assemblerprogrammierung ca. 6,6 μ sec, und bei
- b) Aufruf einer Assemblerroutine von einem Fortranprogramm ca. 100 μ sec benötigt werden.

Der Grund für diese hohen Rechenzeiten liegt im Fall a) in der zeitaufwendigen Handshake-Prozedur, mit der das Interface bedient werden muß. Im Fall b) kommen noch Systemaufrufe hinzu.

Abhilfe kann dadurch geschaffen werden, daß dem externen Prozessor eigene Ein/Ausgabeeinheiten zugeordnet werden, wenn der Universalrechner nur als Steuerrechner fungiert, oder daß der Datenverkehr zwischen beiden Systemen durch ein zusätzliches DMA-Steuerwerk bewerkstelligt wird.

Im letzten Fall entsprechen die Datentransportzeiten der Speicherzykluszeit des langsameren der beiden beteiligten Speicher.

Im folgenden liegt der Schwerpunkt des Interesses bei Systemen, die sich speziell für die Echtzeitverarbeitung eignen.

3. Zerlegbarkeit von Signalverarbeitungsalgorithmen

Primäres Ziel dieses Kapitels ist es, einen gemeinsamen algorithmischen Kern in den hier interessierenden Teilbereichen der digitalen Signalverarbeitung zu finden. Im Mittelpunkt des Interesses steht hier die nichtrekursive sowie die rekursive digitale Filterung, die Berechnung der schnellen Fouriertransformation und die Bildfolgenverarbeitung.

3.1 Signalverarbeitung in Teilprozessoren

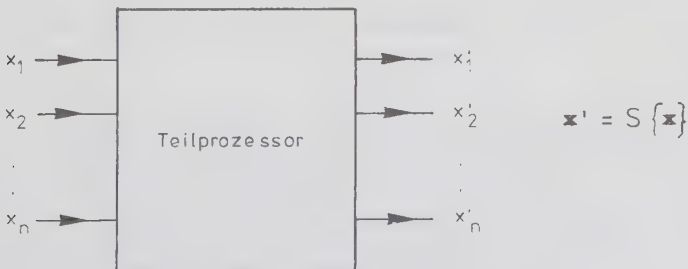
3.1.1 Prinzip

Die Signalverarbeitungsalgorithmen werden im folgenden daraufhin untersucht, wie sie sich in

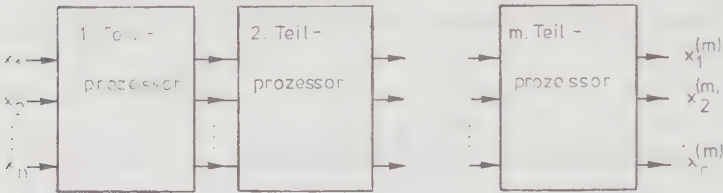
- möglichst kleine (im Sinne einer geringen Zahl von notwendigen Rechenoperationen),
- gleichartige und
- voneinander unabhängige Teilaufgaben

zerlegen lassen.

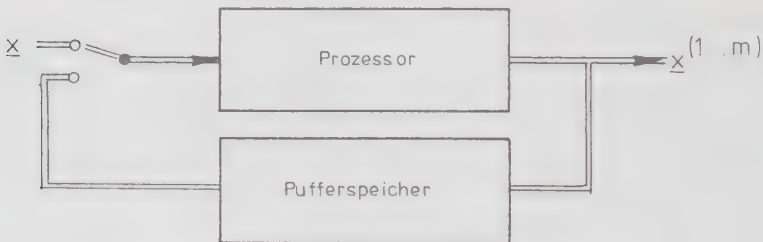
Zur Realisierung dieser Teilaufgaben werden dann Strukturen für "Teilprozessoren" angegeben, die einen Vektor $\mathbf{x}$ von Eingangsgrößen in einem Vektor $\mathbf{x}'$ von Ausgangsgrößen umrechnen. Dabei ist die Dimension des Vektors $\mathbf{x}'$ gleich der von $\mathbf{x}$:



Die Realisierung eines so zerlegten Algorithmus mit Teilprozessoren, die die oben angegebene Eigenschaft bezüglich der Ein- und Ausgangsvektoren aufweisen, wird dann z.B. durch die Kaskadenschaltung einer hinreichend großen Zahl solcher Systeme ermöglicht:



Diese Teilprozessoren können auf verschiedenen Wegen implementiert werden. Z.B. sind sämtliche m Systeme in einem Rechenwerk realisierbar, wenn sein Ausgang auf den Eingang rückgekoppelt wird:



Diese Struktur bedeutet letztlich, daß ein Programm vom Prozessor m Mal durchlaufen wird, wobei die Zwischenergebnisse in Pufferspeicherbereichen übergeben werden. Eine solche Anordnung kann natürlich nur dann in Echtzeit arbeiten, wenn die Rechenzeit zur Bearbeitung des Algorithmus für einen Abtastwert kleiner als ein m -tel des Abtastintervalles ist.

Eine andere Realisierungsmöglichkeit stellt die simultane Bearbeitung der Teilaufgaben dar. Sie läßt sich dadurch erreichen, daß m selbständige, festgekoppelte Prozessoren gleichzeitig an der Ausführung des Algorithmus beteiligt sind. Wenn dabei die Rechenzeit zur Ermittlung eines Ausgangsvektors in jedem Prozessor gleich ist, läßt sich ein wohldefinierter zeitlicher Taktrahmen für den Datenaustausch zwischen den Prozessoren angeben, und es entstehen keine Zeitverluste durch bilaterale Kommunikation benachbarter Prozessoren. Dies ist dann eine Pipelineanordnung.

3.1.2 Rechenzeitabschätzung

Eine grobe Rechenzeitabschätzung für die Ermittlung des Ausgangsvektors $\mathbf{x}'$ aus dem Eingangsvektor $\mathbf{x}$ in einem Teilprozessor läßt sich durch folgenden Ausdruck gewinnen:

$$T_R = (C_M \cdot T_M + C_A \cdot T_A) a$$

mit C_M/C_A = Zahl der Multiplikationen/Additionen,
die er für die Realisierung einer Teilaufgabe auszuführen hat,

T_M/T_A = Rechenzeit für eine Multiplikation/Addition
und $a = 1,5 \dots 2$

Der Faktor a berücksichtigt dabei Datentransportbefehle sowie Schleifenverwaltungs- und Steuerbefehle. Diese Abschätzung ist jedoch nur dann sinnvoll, wenn die Rechenzeiten für die Addition bzw. die Multiplikation erheblich (z.B. um den Faktor 5 bis 10) über den Datentransport- oder Steuerbefehlszeiten liegt. Diese Voraussetzung ist z.B. bei den meisten derzeitigen Mikroprozessoren mit festem Befehlssatz gegeben. In vielen neueren Konzepten für Signalprozessoren ist dies jedoch nicht der Fall, weil die Multiplikation, die Addition oder auch ein Datentransport in nur einem Maschinenzklus bearbeitet werden kann. Dadurch kann der Faktor a wesentlich größer werden, als oben angegeben ist.

Dies zu vermeiden, ist eines der Hauptziele beim Entwurf der Architektur eines Signalprozessors oder seiner Teilprozessoren.

Es wird erreicht durch geschickte Wahl der Datenpfade und eine möglichst weitgehende Automatisierung der Schleifenverwaltung.

In den nachfolgenden Abschnitten werden Strukturen solcher Teilprozessoren ermittelt.

3.2 Algorithmenbeispiele

3.2.1 Nichtrekursive digitale Filter

Nichtrekursive digitale Filter sind in ihrer Struktur ein Spezialfall der rekursiven Filter. Da sie in ihrem Aufbau jedoch einfacher sind, werden sie vor den letztgenannten betrachtet. Dabei wird ihre Implementierung in einem Prozessor vergleichsweise ausführlich behandelt, da sie auch für den späteren Abschnitt 3.2.5 von Bedeutung sind.

3.2.1.1 Filtergleichung

Der Ausgangswert $y(k)$ eines nichtrekursiven Filters ergibt sich aus

$$y(k) = \sum_{\mu=0}^n h_0(\mu) u(k-\mu) = h_0(k) * u(k) \quad (3.2-1)$$

Dabei ist $h_0(k)$ = Impulsantwort
 n = Filtergrad.

Zur Berechnung eines Wertes $y(k)$ sind $n+1$ Werte der Eingangsfolge $u(k)$ erforderlich. Zu deren Abspeicherung genügt also ein endliches Datenfeld der Länge $n+1$ und ein Feld gleicher Länge für die Impulsantwort.

Da nichtrekursive Systeme stets stabil sind, erübrigen sich Stabilisierungsmaßnahmen, wie sie z.B. bei der Realisierung

rekursiver Filter nötig sind. Dagegen ist möglicherweise eine Skalierung vorzunehmen [1].

Im folgenden wird die direkte Realisierung von Gl. (3.2-1) behandelt, nicht dagegen die nichtrekursive Filterung durch die sogenannte "schnelle Faltung" (z.B. [1]).

3.2.1.2 Sonderfälle

a) linearphasige Filter

Eine spezielle Klasse nichtrekursiver Filter weist einen streng linearen Phasengang auf. Hier interessiert, daß ihre Impulsantwort symmetrisch bzw. antimetrisch ist:

$$h_0(\mu) = \pm h_0(n-\mu) \quad 0 \leq \mu < \frac{n}{2} \quad . \quad (3.2-2)$$

Es ergeben sich Vorteile für den Algorithmus, wenn auf eine geringe Zahl von Multiplikationen Wert gelegt wird. Setzt man Gl. (3.2-2) in Gl. (3.2-1) ein, erhält man

$$y(k) = \sum_{\mu=0}^{N-1} h_0(\mu) \{u(k-\mu) \pm u(k-n+\mu)\} + h_0\left(\frac{n}{2}\right) u\left(k - \frac{n}{2}\right) \\ \text{für } n = 2N, N \in \mathbb{N} \quad . \quad (3.2-3)$$

Im Fall eines ungeradzahligen Filtergrades ergibt sich

$$y(k) = \sum_{\mu=0}^N h_0(\mu) \{u(k-\mu) + u(k-n+\mu)\} \\ \text{für } n = 2N+1, N \in \mathbb{N}_0 \quad . \quad (3.2-4)$$

Die notwendigen arithmetischen Operationen zur Ausführung der unterschiedlichen Filtertypen zeigt folgende Tabelle. Aus ihr geht hervor, daß linearphasige Filter im wesentlichen mit der halben Anzahl der im allgemeinen Fall nötigen Anzahl von Multiplikationen auskommen.

	Gl. 3.2-1	Gl. 3.2-3	Gl. 3.2-4
Additionen	n	n	n
Multiplikationen	n+1	$\frac{n}{2} + 1$	$\frac{n+1}{2}$

b) Korrelation

Die Definitionsgleichung für die Korrelationsfunktion hat in ihrer Struktur große Ähnlichkeit mit Gleichung (3.2-1) für die nichtrekursive Filterung. Deshalb wird sie hier kurz beschrieben.

Die Autokorrelationsfunktion wird erklärt durch [14]:

$$\varphi_{xx}(\kappa) = \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{k=-N}^N x(k)x(\kappa+k) , \quad (3.2-5a)$$

entsprechend die Kreuzkorrelation:

$$\varphi_{xy}(\kappa) = \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{k=-N}^N x(k)y(\kappa+k) . \quad (3.2-5b)$$

Wegen des in Gleichung 3.2-5a/b notwendigen Grenzwertprozesses, der die Berücksichtigung unendlich vieler Abtastwerte der Folgen $x(k)$ bzw. $y(k)$ fordert, sind die angegebenen Ausdrücke nicht allgemein realisierbar. Führt man die Summation nur über endlich viele Werte aus, so erhält man einen Schätzwert der Korrelierten. Auf die Frage, wie groß N zu wählen ist, bzw. wieviele verschiedene Werte κ verwendet werden dürfen, wird hier nicht eingegangen. In Gleichung (3.2-5a/b) ist der Term

$$\frac{1}{2N+1}$$

notwendig zur Sicherstellung der Konvergenz der Reihe bei stationären Folgen. Bei der Summation über endlich viele Glieder ist er lediglich ein Normierungsfaktor, mit dem ein Überlauf im Rechenwerk des Prozessors vermieden wird.

Mit

$$\tilde{x}(k) = \frac{1}{\sqrt{2N+1}} x(k)$$

$$\tilde{y}(k) = \frac{1}{\sqrt{2N+1}} y(k)$$

erhält man aus Gl. (3.2-5a/b) dann den Schätzwert der Korrelierten

$$\tilde{\varphi}_{xx}(\kappa) = \sum_{k=-N}^{+N} \tilde{x}(k) \tilde{x}(\kappa+k) \quad (3.2-6a)$$

$$\tilde{\varphi}_{xy}(\kappa) = \sum_{k=-N}^{+N} \tilde{x}(k) \tilde{y}(\kappa+k) \quad . \quad (3.2-6b)$$

Führt man nun noch die Substitution

$$k = v-N$$

ein, ergibt sich mit $x'(v) = \tilde{x}(v-N)$ und $y'(v) = \tilde{y}(v-N)$ für die Korrelationsfunktion

$$\tilde{\varphi}_{xx}(\kappa) = \sum_{v=0}^{2N} x'(v) x'(v+\kappa) \quad (3.2-7a)$$

$$\text{bzw. } \tilde{\varphi}_{xy}(\kappa) = \sum_{v=0}^{2N} x'(v) y'(v+\kappa) \quad . \quad (3.2-7b)$$

Die in Gleichung 3.2-7a angegebene Rechenvorschrift für die Autokorrelationsfunktion unterscheidet sich von einem nichtrekursiven Filter im wesentlichen dadurch, daß anstelle der konstanten Koeffizienten $h_0(v)$ ein Segment des Signales $x'(v)$ selbst verwendet wird.

Der Ausdruck 3.2-7b zur Berechnung der Kreuzkorrelierten entspricht direkt einem nichtrekursiven Filter.

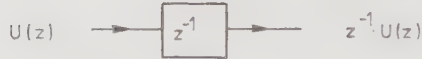
3.2.1.3 Darstellung im z-Bereich

Zur einfacheren formalen Handhabung der Filtergleichung wird (3.2-1) mittels einer Übertragungsfunktion $H(z)$ im z-Bereich dargestellt [1].

Dabei geht eine Signalverzögerung um ein Abtastintervall



über in



wobei $U(z)$ die Z-Transformierte der Eingangsfolge $u(k)$ ist:

$$U(z) = Z\{u(k)\} = \sum_{k=0}^{\infty} u(k) z^{-k} \quad (3.2-8)$$

Gleichung (3.2-1) wird dann zu

$$Y(z) = H(z) \cdot U(z) \quad (3.2-9)$$

mit der Übertragungsfunktion

$$H(z) = z^{-n} \sum_{\mu=0}^n h_0(\mu) z^{n-\mu} \quad (3.2-10)$$

Bild (3.2-1) zeigt den durch Gleichung (3.2-9) angegebenen Zusammenhang.

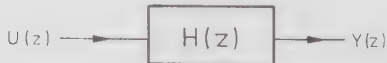


Bild (3.2-1) Zum Signalfluß im Z-Bereich

3.2.1.4 Realisierung der Filtergleichung mit Teilprozessoren

Ziel des folgenden ist es, die in Gleichung (3.2-9) und (3.2-10) vorkommenden Rechenoperationen in möglichst kleine, voneinander unabhängige Teiloperationen aufzuspalten und so eine simultane Bearbeitung zu ermöglichen. Dabei muß die Aufspaltung so vorgenommen werden, daß die Operationen zeitlich entkoppelt sind, d.h. sie müssen gleichzeitig durch mehrere Prozessoren ausführbar sein. Geht man davon aus, daß bei einem nichtrekursiven Digitalfilter das Eingangssignal $U(z)$ sowie ein Zwischenergebnis von Prozessor zu Prozessor weitergegeben werden muß, kann man folgenden Ansatz machen:

Ein Prozessor hat jeweils zwei Ein- und Ausgangskanäle

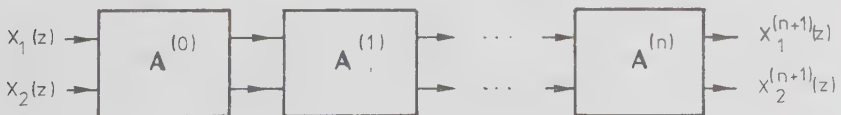


wobei sein Verhalten durch eine Matrix $\mathbf{A}$ beschrieben wird:

$$\mathbf{x}' = \mathbf{A} \mathbf{x} \quad (3.2-11)$$

Die Matrix $\mathbf{A}$ beschreibt im folgenden die Wirkung eines Teilprozessors auf einen Eingangsvektor $\mathbf{x}$, dessen Elemente z -Transformierte sind.

Werden $n+1$ Prozessoren in Kaskade geschaltet, erhält man



und

$$\mathbf{x}^{(n+1)} = \mathbf{A}^{(n)} \cdot \mathbf{A}^{(n-1)} \dots \mathbf{A}^{(0)} \mathbf{x} = \prod_{\mu=0}^n \mathbf{A}^{(\mu)} \cdot \mathbf{x} \quad (3.2-12)$$

Für die Matrix $\mathbf{A}^{(\mu)}$ wird eine untere Dreiecksmatrix der folgenden Form gewählt:

$$\mathbf{A}^{(\mu)} = \begin{bmatrix} a_{11} & 0 \\ a_{21}^{(\mu)} & a_{22} \end{bmatrix}.$$

Wie durch die Bezeichnung angedeutet, seien hier die a_{11} , a_{22} konstant, während a_{21} von μ abhängig ist. Dann ergibt sich für das Produkt der Matrizen $\mathbf{A}^{(\mu)}$ der Ausdruck:

$$\prod_{\mu=n}^0 \mathbf{A}^{(\mu)} = \begin{bmatrix} a_{11}^{n+1} & 0 \\ \sum_{\mu=0}^n a_{22}^{n-\mu} \cdot a_{11}^{\mu} \cdot a_{21}^{(\mu)} & a_{22}^{n+1} \end{bmatrix}. \quad (3.2-13)$$

Wählt man $a_{11} = z^{-1}$, $a_{21}^{(\mu)} = h_0^{(\mu)}$ und $a_{22} = 1$, so liefert Gleichung (3.2-12) mit (3.2-13)

$$\mathbf{X}^{(n+1)} = \begin{bmatrix} z^{-(n+1)} & 0 \\ \sum_{\mu=0}^n z^{-\mu} h_0^{(\mu)} & 1 \end{bmatrix} \mathbf{X}. \quad (3.2-14)$$

Mit

$$\mathbf{X} = \begin{bmatrix} U(z) \\ 0 \end{bmatrix}$$

folgt dann

$$\begin{aligned} X_1^{(n+1)} &= z^{-(n+1)} U(z) \\ X_2^{(n+1)} &= U(z) \sum_{\mu=0}^n z^{-\mu} h_0^{(\mu)} \\ &= U(z) \cdot z^{-n} \sum_{\mu=0}^n h_0^{(\mu)} z^{n-\mu} \\ &= U(z) \cdot H(z) = Y(z) \end{aligned}$$

Das Ausgangssignal X_2 des letzten Prozessors liefert also das gesuchte, nichtrekursiv gefilterte Signal.

Die Struktur eines durch

$$\mathbf{A}^{(\mu)} = \begin{bmatrix} z^{-1} & 0 \\ h_0(\mu) & 1 \end{bmatrix}$$

beschriebenen Prozessors zeigt Bild (3.2-2). Fragen der Realisierung eines solchen Bausteins werden in Abschnitt 3.2-7 behandelt.

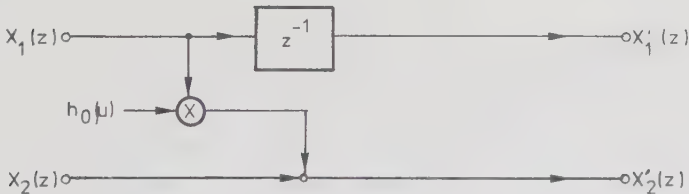


Bild (3.2-2) Struktur eines Teilprozessors für die nichtrekursive Filterung

Die Struktur eines Filters n-ten Grades zeigt Bild (3.2-3).

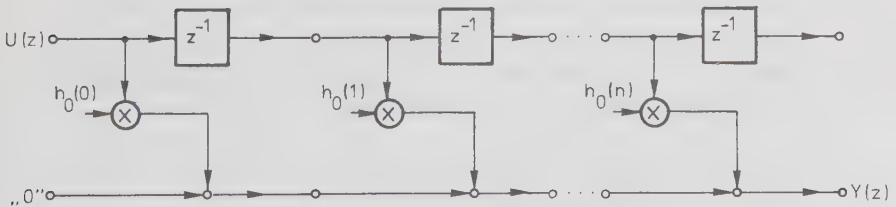


Bild (3.2-3) Struktur eines Filters n-ten Grades mit $n+1$ Teilprozessoren

3.2.1.5 Zur Programmierung des nichtrekursiven Filters

Es sollen nun Programmbeispiele für die Realisierung eines nichtrekursiven Filters nach Bild (3.2-3) angegeben werden, wenn sämtliche Teilprozessoren auf einem sequentiell arbeitenden Prozessor programmiert werden.

a) Schleifenfreie Programmierung

Bei der schleifenfreien Programmierung des Algorithmus wird jeder Teilprozessor durch einen eigenen Programmblock realisiert, d.h. es werden keine Programmteile mehrfach genutzt. Durch den Wegfall der Schleifenverwaltung werden so die höchstmöglichen Geschwindigkeiten erreicht.

Der v -te innere Programmblock besteht dann z.B. aus folgenden Maschinenbefehlen:

```
u(k-n+v)  → MUL ; Laden des Multiplizierers
h0(n-v)   → MUL ; Laden des Multiplizierers
Multipliziere
Produkt    → Akku
Addiere Produkt zum Akkuinhalt
u(k-n+v)   → u(k-n+v-1) ; Weiterschieben der
                                   Zustandsvariablen.
```

Hierbei sind die Variablen $u(k)$ in fest vereinbarten Speicherplätzen des Arbeitsspeichers abgelegt.

Das gesamte Filterprogramm ist dann realisierbar durch Kaskadenschaltung von Programmblöcken:

```
BEG:   Einlesen eines neuen Abtastwertes
        1. Programmblock
        2. Programmblock
        :
        :
        (n+1)-ter Programmblock
        Ausgabe eines Ausgangswertes
        Rücksprung nach BEG      .
```

Die genaue Analyse dieses Programms ergibt, daß zu seiner Ausführung $4(n+1)+2$ Transport- und Steuerbefehle notwendig sind, sowie $(n+1)$ Multiplikationen und Additionen.

Eine Abschätzung der Rechenzeit für einen Filterzyklus ergibt sich somit aus

$$T_R = (n+1)(T_M + T_A) + [4(n+1)+2]T_{TR} \quad (3.2-15)$$

mit T_M = Multiplikationszeit

T_A = Additionszeit

T_{TR} = Datentransportzeit .

Es wird hierbei nicht unterschieden zwischen den Rechenzeiten für Steuer- und Datentransportbefehle. Als Beispiel sei der MOS-Mikroprozessor TMS 9900 erwähnt, der 16 Bit parallel verarbeitet und einen festverdrahteten Multiplikationsbefehl zur Verfügung stellt. Für ihn erhält man nach [15]:

$$T_R = (n+1) 37 \mu\text{sec} + [4(n+1)+2] \cdot 5 \mu\text{sec}$$

Für $n = 10$ ergibt sich $T_R = 637 \mu\text{sec}$.

Die tatsächliche Rechenzeit beträgt gegenüber dieser Abschätzung $571 \mu\text{sec}$. Der Grund für die Differenz liegt darin, daß die Zahl der notwendigen Transportbefehle geringer als in (3.2-15) angegeben ist, weil ein Teil der Operanden in den internen Rechenregistern des Prozessors gehalten werden kann und deshalb nicht laufend aus dem Speicher geholt werden muß.

b) Indizierung

Das zuletzt angegebene schleifenfreie Programm für ein nichtrekursives Digitalfilter kann bei allen Datentransportbefehlen fest vereinbarte absolute Adressen verwenden, da kein Programmteil mehrfach verwendet wird.

Dieses Vorgehen ist dann sinnvoll, wenn die maximal mögliche Rechengeschwindigkeit durch Einsparung der Schleifenverwaltung erreicht werden soll. Bei größeren Filterlängen ergeben sich jedoch sehr lange Programme, so daß die Verwendung von Programmschleifen wünschenswert erscheint.

Im angegebenen Programmbeispiel würde dies bedeuten, daß der Programmblock n-1 mal gerechnet wird, wobei sich jedesmal die Adressen ändern müssen. Bei geeigneter Adressierung lassen sich dabei die Befehle

$$u(k-n+v) \rightarrow u(k-n+v-1)$$

einsparen, die das Weiterschieben der Zustandsvariablen $u(k)$ im Speicher bewirken. Die hierzu notwendigen Operationen zur Speicherindizierung werden im folgenden ausführlich angegeben, da das Weiterschieben von Daten eine in der Signalverarbeitung häufig wiederkehrende Operation ist.

Gemäß Gleichung (3.2-1) müssen n+1 aufeinanderfolgende Werte der Eingangsfolge $u(k)$ für die Berechnung eines Ausgangswertes $y(k)$ herangezogen werden. Zur Abspeicherung n vergangener und des aktuellen Wertes von $u(k)$ muß eine Transformation des Zeitparameters k in den Ortsparameter κ vorgenommen werden, wobei κ die Adresse ist, unter der der Prozessor Zugriff auf den Wert $u(k)$ hat.

Diese Transformation wird realisiert durch

$$\kappa = k \Big|_{\text{mod}(n+1)} \quad . \quad (3.2-16)$$

Damit erhält man aus Gleichung (3.2-1)

$$y(k) = \sum_{\mu=0}^n h_0(\mu) \quad u((k-\mu) \Big|_{\text{mod}(n+1)}) \quad . \quad (3.2-17)$$

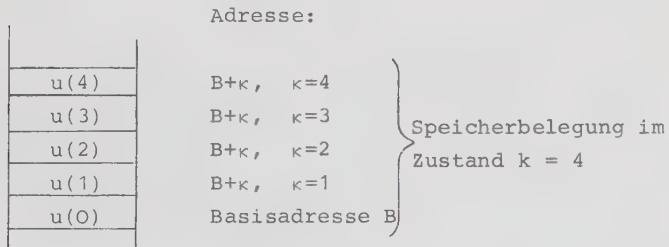
Führt man in (3.2-17) die Substitution $\ell = n-\mu$ ein, so ergibt sich

$$y(k) = \sum_{\ell=0}^n h'_0(\ell) \cdot u((k-n+\ell) \bmod (n+1)) \quad (3.2-18)$$

mit $h'_0(\ell) = h_0(n-\ell)$.

Der Vorteil dieser Darstellung liegt darin, daß bei der Programmierung von Gleichung (3.2-18) der Summationsindex ℓ nur inkrementiert zu werden braucht. Dies wird anhand eines Diagramms und eines Programmbeispiels noch näher beschrieben.

Die Anordnung der Werte $u(k)$ im Datenspeicher des Prozessors wird üblicherweise durch eine lineare Speicherorganisation beschrieben, wie dies das folgende Diagramm für $n = 4$ zeigt:



Hiervon ausgehend, läßt sich exemplarisch eine Programmskizze für ein nichtrekursives Filter, das Gleichung (3.2-18) erfüllt, direkt angeben:

κ: = 0 ; Löschen des Indexzählers

Akku: = 0 ; Löschen des Akkus

BEG: u(k) → u(κ) ; Transport eines Eingangswertes in den Speicher mit der Adresse κ = k $\bmod (n+1)$

ℓ: = -1 ; Setzen des Schleifenzählers

$S: \ell := \ell + 1$; Schleifenbeginn
 $\kappa := \kappa + 1 \mid_{\text{mod}(n+1)}$; Indexrechnung
 $h_o'(\ell) \rightarrow \text{MUL}$; Transport des ℓ -ten Koeffizienten
in dem Multiplizierer
 $u(\kappa) \rightarrow \text{MUL}$; Transport der κ -ten Zustandsvariable
in den Multiplizierer
MUL ; Multiplikation
MUL $\rightarrow$ Akku ; Transport des Produktes zum Akku
ADD ; Akkumulation
Falls $(\ell - n) < 0$, Sprung nach S; Abfrage auf Schleifenende

Akku $\rightarrow$ Ausgabe ; Ausgabe eines Wertes $y(k)$
Akku := 0 ; Löschen des Akkus
 $\kappa := \kappa + 1 \mid_{\text{mod}(n+1)}$; Indexrechnung
Springe nach BEG ; Rücksprung.

Für die Abschätzung der Rechenzeit wird zur Realisierung der Operation $\kappa := (\kappa + 1) \mid_{\text{mod}(n+1)}$ ein Additions-, ein Sprung- und zwei Datentransportbefehle vorgesehen. Damit sind in der Schleife

	1	Multiplikation
	3	Additionen
	5	Datentransporte
und	2	Sprungbefehle

zu realisieren. Dies führt gegenüber der Realisierung des zuvor angegebenen Programms zu einem zeitlichen Mehraufwand von

$$\Delta T_R = [3(n+1)+3]T_{TR} + 2(n+1)T_{ADD} \quad (3.2-19)$$

wenn die Ausführungszeiten für Sprung-, Steuer- und Datentransportbefehle als gleich angesetzt werden. Für den Mikroprozessor TMS 9900 bedeutet dies für $n = 10$ eine zusätzliche Laufzeit von ca. 290 μsec .

3.2.2 Rekursive Digitalfilter

In einem rekursiven Digitalfilter ergibt sich ein Ausgangswert $y(k)$ aus

$$y(k) = \sum_{\mu=0}^m b_{m-\mu} u(k-\mu) - \sum_{v=1}^n c_{n-v} y(k-v) \quad (3.2-20)$$

Hierbei ist n der Filtergrad, aus Kausalitätsgründen muß $n \geq m$ sein. Der wesentliche Unterschied zum nichtrekursiven Filter liegt in der Mitverwendung vergangener Ausgangswerte $y(k)$. Durch z -Transformation der Ein- und Ausgangsfolge erhält man

$$Y(z) = U(z) \cdot \frac{\sum_{\mu=0}^m b_{m-\mu} z^{-\mu}}{1 + \sum_{v=1}^n c_{n-v} z^{-v}} = U(z) \cdot H(z) \quad (3.2-21)$$

$H(z)$ = Übertragungsfunktion

Für die folgende Darstellung wird $H(z)$ geschrieben als

$$H(z) = \prod_{\lambda=1}^{\ell} H_{\lambda}(z) \quad (3.2-22)$$

mit

$$H_{\lambda}(z) = \frac{b_{2\lambda} z^2 + b_{1\lambda} z + b_{0\lambda}}{z^2 + c_{1\lambda} z + c_{0\lambda}}$$

wobei gilt:

$$\ell = \frac{n}{2} \quad , \quad \text{wenn } n \text{ gerade}$$

und

$$\ell = \frac{n-1}{2} \quad \text{und } b_{0\lambda} = c_{0\lambda} = 0 \text{ für ein } \lambda, \\ \text{wenn } n \text{ ungerade.}$$

Dies entspricht einer Kaskadenschaltung von Blöcken 2. Grades. Sie hat eine Reihe von praktischen Vorteilen, die hier jedoch nicht näher erläutert werden (z.B. [1, 16]).

Bild (3.2-4) zeigt die Anordnung bei der Serienschaltung von ℓ Blöcken .

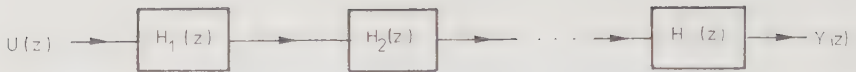


Bild (3.2-4) Blockschaltbild eines rekursiven Filters 2ℓ -ten Grades (bzw. $(2\ell-1)$ -ten Grades)

In Gleichung (3.2-22) lassen sich die Nennerterme der Übertragungsfunktionen $H_\lambda(z)$ auch darstellen durch ihre Pole:

$$z^2 + c_{1\lambda}z + c_{0\lambda} = (z - z_{\infty\lambda})(z - z_{\infty\lambda}^*)$$

$$\text{bzw.} \quad z + c_{1\lambda} = z - z_{\infty\lambda} \quad (3.2-23)$$

Aus Stabilitätsgründen wird an die Pole $z_{\infty\lambda}$ die Forderung gestellt, daß

$$|z_{\infty\lambda}| < 1 \quad \forall \lambda$$

gilt. Dies führt für die Koeffizienten $c_{v\lambda}$ in Gleichung (3.2-23) zu der Einschränkung

$$|c_{0\lambda}| < 1$$

$$\text{und} \quad |c_{1\lambda}| < 1 + c_0$$

Somit kann der Koeffizient $c_{1\lambda}$ im Bereich

$$-2 < c_{1\lambda} < +2$$

liegen.

3.2.2.1 Beispiel für die Struktur eines rekursiven Blockes 2. Grades

Aus Gleichung (3.2-20) ergibt sich für $n = 2$

$$y(k) = b_2u(k) + b_1u(k-1) + b_0u(k-2) - c_1y(k-1) - c_0y(k-2) \quad (3.2-24)$$

Bild (3.2-5) zeigt eine realisierende Struktur. Die Zustandsvariablen $u(k-\mu)$ und $y(k-\nu)$ werden in den Speicherplätzen gemäß der eingezeichneten Pfeilrichtung nach jedem Rechenzyklus verschoben.

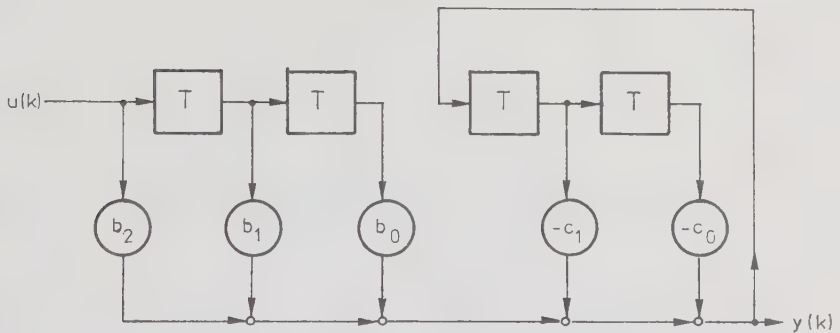


Bild (3.2-5) Blockdiagramm zur Realisierung eines rekursiven Filterblockes 2. Grades

Bei der Kaskadierung mehrerer Blöcke 2. Grades wie sie in Bild (3.2-5) angegeben ist, ist es nicht notwendig, für jeden Block 4 Speicherplätze zu investieren. Vielmehr lassen sich die Speicherzellen für $y(k-1)$ und $y(k-2)$ für den folgenden Block mitverwenden, wie aus Bild (3.2-6) hervorgeht. Es werden dann $n+2$ Zustandsvariablenpeicher benötigt, d.h. die Struktur ist nicht kanonisch.

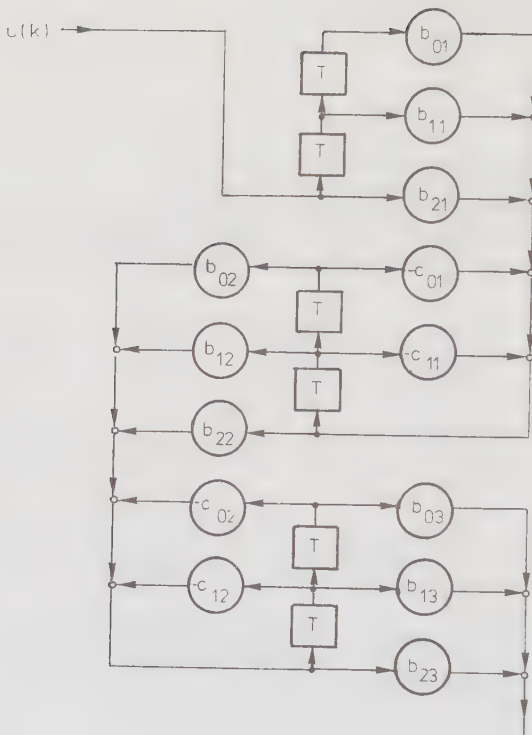


Bild (3.2-6) Kaskadierung digitaler Filterblöcke 2. Grades

3.2.2.2 Probleme bei Festkommarealisierung

Die Realisierung digitaler Filter in Festkommaarithmetik ist immer dann sinnvoll, wenn hohe Rechengeschwindigkeiten nötig sind, oder ein zu realisierender Prozessor aus Aufwandsgründen nicht mit einem Gleitkommaeichenwerk ausgestattet werden kann. Durch das Rechnen mit Festkommazahlen entstehen eine Reihe von Problemen, die allerdings bei Verwendung der Gleitkommatechnik auch nicht immer beseitigt werden können.

Von den auftretenden Problemen werden hier nur diejenigen betrachtet, die Einfluß auf die Realisierung eines rekursiven Digitalfilters in einem Prozessor nehmen. Im übrigen wird auf [1] und [16] verwiesen.

In der digitalen Signalverarbeitung wird üblicherweise mit Zweierkomplementzahlen gerechnet, eine Zahl x wird dabei durch die Ziffernfolge z_i gemäß

$$x = -z_0 + \sum_{i=1}^{w-1} z_i 2^{-i} \quad , \quad z_i \in \{0,1\} \quad (3.2-25)$$

w = Wortlänge

beschrieben. Dadurch kann sie nur im Bereich

$$-1 \leq x < +1$$

dargestellt werden.

Werden mehrere 2'er-Komplementzahlen aufaddiert, so daß die Summe außerhalb des möglichen Bereichs liegt, so wird sie durch die in Bild (3.2-7) dargestellte Überlaufcharakteristik auf den angegebenen Bereich begrenzt.

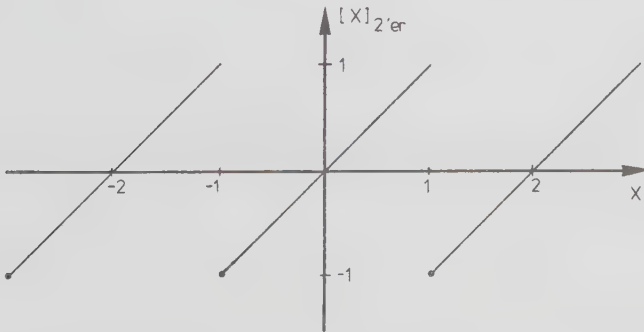


Bild (3.2-7) Überlaufcharakteristik des 2'er-Komplements

Bei der Multiplikation zweier n -stelliger Zahlen entsteht i.a. ein $2n$ -stelliges Produkt, das durch eine nichtlineare Operation (Abschneiden, Runden) zu einem n -stelligen Zahlenwort gekürzt werden muß.

Die durch die endliche Wörtlänge auftretenden, hier interessierenden Probleme werden kurz erläutert.

a) Grenzzyklen

"Kleine Grenzzyklen" können in rekursiven Systemen durch die nichtlineare Operation der Wortlängenverkürzung nach der Multiplikation entstehen [1]. Sie können meist dadurch beseitigt werden, daß das Rundungsbit nicht in Abhängigkeit der niedrigwertigen Hälfte des doppelten Produktes gesetzt wird, sondern einem binären Zufallsgenerator entnommen wird [17].

"Große Grenzzyklen" können durch die nichtlineare Überlaufkennlinie (Bild (3.2-7)) des 2'er-Komplements entstehen. Sie sind (unter gewissen weiteren Randbedingungen) z.B. dadurch vermeidbar, daß eine Überlaufkennlinie nach Bild (3.2-8) Verwendung findet [1].

Die Softwarerealisierung der angegebenen Rundungsstrategie und Überlaufkennlinie ist sehr zeitaufwendig, so daß in einem Signalprozessor für beide Verfahren eine Hardwarelösung anzustreben ist.

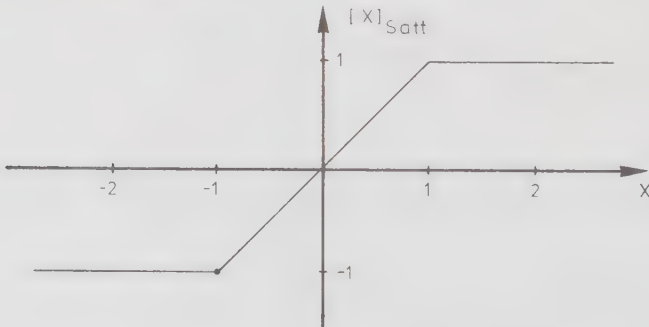


Bild (3.2-8) Sättigungskennlinie zur Beseitigung von
"Großen Grenzzyklen"

b) Koeffizientennormierung

b.1) Zählerkoeffizienten

Der Entwurf eines Filters liefert häufig für die Koeffizienten $b_{\mu\lambda}$ Zahlen, die betragsmäßig größer als Eins und somit nicht darstellbar sind.

Es ist deshalb notwendig, die einzelnen Filterblöcke nach Gleichung (3.2-22) auf den größten vorkommenden Koeffizienten zu normieren:

$$H_{\lambda}(z) = \frac{b_{2\lambda}z^2 + b_{1\lambda}z + b_{0\lambda}}{z^2 + c_{1\lambda}z + c_{0\lambda}} = b_{N\lambda} \frac{\frac{b_{2\lambda}}{b_{N\lambda}}z^2 + \frac{b_{1\lambda}}{b_{N\lambda}}z + \frac{b_{0\lambda}}{b_{N\lambda}}}{z^2 + c_{1\lambda}z + c_{0\lambda}} \quad (3.2-26)$$

$$\text{mit } b_{N\lambda} = \text{MAX}\{|b_{2\lambda}|, |b_{1\lambda}|, |b_{0\lambda}|\} + 2^{-(w-1)}.$$

Bei der Realisierung der Filtergleichung verzichtet man dann auf die Multiplikation mit $b_{N\lambda}$, so daß eine Übertragungsfunktion

$$H'(z) = \prod_{\lambda=1}^L \frac{1}{b_{N\lambda}} \cdot H_{\lambda}(z) \quad (3.2-27)$$

resultiert.

Dies bedeutet u.U. eine erhebliche Einbuße im Dynamikbereich des Filters. Wenn dies nicht toleriert werden kann, muß entweder mit höherer innerer Wortlänge gerechnet, oder durch einen Optimierungsprozeß eine geeignete Skalierung der Filterkoeffizienten gesucht werden [16].

b.2) Nennerkoeffizienten

Wie zu Beginn dieses Abschnittes erklärt wurde, kann auch der Koeffizient $c_{1\lambda}$ außerhalb des im 2'er Komplexbereich zulässigen Zahlenbereiches liegen: $|c_{1\lambda}| > 2$. Zu seiner Realisierung werden hier zwei Methoden angegeben.

- Aus Gleichung (3.2-24) gewinnt man

$$\frac{1}{2} y(k) = \frac{b_2}{2} u(k) + \frac{b_1}{2} u(k-1) + \frac{b_0}{2} u(k-2) + \\ - \frac{c_1}{2} y(k-1) - \frac{c_0}{2} y(k-2)$$

wobei $|c_1/2|$ und $|c_0/2|$ kleiner Eins sind. Mit dieser Normierung wird das Ergebnis $y(k)$ um den Faktor 2 zu klein, es muß also um eine Bitposition in Richtung zum höchstwertigen Bit geschoben werden, bevor es in die Zustandsvariablen Speicher transportiert wird. In einem speziellen Rechenwerk eines Signalprozessors kann diese Operation festverdrahtet sein. Der Nachteil dieses Verfahrens ist der Verlust eines Bits für die Koeffizientendarstellung.

- Eine zweite Möglichkeit zur Realisierung des Koeffizienten c_1 ist darin zu sehen, daß ein Filterblock (Bild (3.2-5,-6)) mit 6 Multiplikationen gerechnet wird, wobei zweimal der Koeffizient $c_1/2$ als Faktor dient [18]. Wegen $-1 < \frac{c_1}{2} < 1$ darf dann c_1 im Intervall $[-2 \dots +2] (1\text{-LSB})$ liegen.

3.2.2.3 Teilprozessorstruktur

Es soll im folgenden entsprechend dem Vorgehen bei nichtrekursiven Filtern eine Teilprozessorstruktur angegeben und analysiert werden, die zu einer einfachen Kopplung von Einzelprozessoren führt.

Dazu wird in Anlehnung an Bild (3.2-2) ein Prozessor eingeführt, der einen bidirektionalen Datenfluß erlaubt. Seine Struktur zeigt Bild (3.2-9a).

Er wird beschrieben durch die Übertragungsmatrix $\mathbf{A}^{(v)}$

$$\mathbf{x}' = \mathbf{A}^{(v)} \mathbf{x}, \quad \mathbf{A}^{(v)} = \begin{bmatrix} z & 0 \\ -c_v & 1 \end{bmatrix} \quad (3.2-28)$$

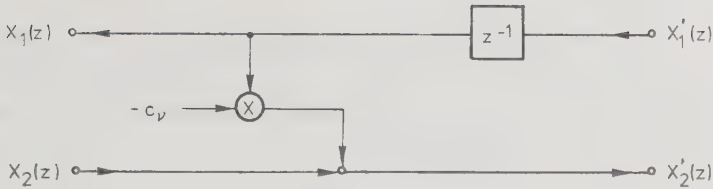


Bild (3.2-9a) Teilprozessor mit bidirektionalem Datenfluß

Es interessiert hier die Hintereinanderschaltung zweier durch Gleichung (3.2-28) beschriebener Prozessoren. Die Kaskadenschaltung ergibt per Schaltungszwang, daß $X_1'(z)$ und $X_2'(z)$ die Eingangsgrößen des nächsten Systems sind. Für dessen Ausgangsgrößen gilt dann

$$\mathbf{X}'' = \left(\begin{matrix} \text{O} \\ \prod_{v=1} \mathbf{A}^{(v)} \end{matrix} \right) \mathbf{X} = \begin{bmatrix} z^2 & 0 \\ -zc_1 - c_0 & 1 \end{bmatrix} \mathbf{X} \quad (3.2-29)$$

Um ein rekursives System zu erhalten, wird die Ausgangsgröße X_2'' als Eingangsgröße X_1' verwendet, d.h. $X_2'' = X_1'$ und man erhält mit Gleichung (3.2-29)

$$X_1 z^2 = X_2''$$

$$(-zc_1 - c_0)X_1 + X_2 = X_2'' \quad .$$

Durch Elimination von X_1 folgt

$$X_2'' = X_2 \frac{z^2}{z^2 + c_1 z + c_0} \quad . \quad (3.2-30)$$

Dies entspricht dem rekursiven Teil der Übertragungsfunktion - multipliziert mit z^2 - eines Filterblockes 2. Grades, wie er durch Gleichung (3.2-22) beschrieben wird (Bild (3.2-9b)). Zur Realisierung des nichtrekursiven Teils können drei Teilprozessoren nach Bild (3.2-2) Verwendung finden, deren Übertragungsmatrizen die Form

$$\mathbf{A}^{(\nu)'} = \begin{bmatrix} z^{-1} & 0 \\ b_{m-\mu} & 1 \end{bmatrix}$$

haben (hier speziell: $m=2$).

Die Zusammenschaltung dieser Teilrechenwerke zu einem Filter 2. Grades zeigt Bild (3.2-10). Der Vorteil dieser Anordnung liegt in der außerordentlich einfachen Programmierung und Synchronisation der Rechenwerke.



Bild (3.2-9b) Rekursiver Teil des Filters 2. Grades

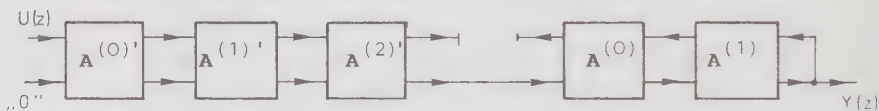


Bild (3.2-10) Rekursives digitales Filter 2. Grades,
realisiert durch 5 simultan arbeitende Prozessoren

Bei gegebener Zahl von Prozessoren lassen sich höhere Filtergrade durch mehrfaches Durchlaufen der Signalwerte durch die Prozessorkette erreichen [19]. Dadurch verringert sich natürlich die maximal mögliche Durchsatzrate. Es sei hier angemerkt, daß auch diese Anordnung bezüglich der Speicher nicht kanonisch ist.

3.2.3 Die schnelle Fouriertransformation (FFT)

Die diskrete Fouriertransformation (DFT) wird erklärt durch (z.B. [14])

$$\text{DFT}(u(k)) = U(\mu) = \sum_{k=0}^{M-1} u(k) W_M^{k\mu}, \quad (3.2-31)$$

mit

$$w_M^{k\mu} = e^{-j(\frac{2\pi}{M} k\mu)} = \cos \frac{2\pi}{M} k\mu - j \sin \frac{2\pi}{M} k\mu$$

und

M = Transformationslänge.

Die "schnelle Fouriertransformation" hat zum Ziel, den in Gl. (3.2-31) angegebenen Ausdruck möglichst ökonomisch zu berechnen. Dies wird erreicht durch eine Aufspaltung der DFT derart, daß mehrere geeignete Fouriertransformationen kürzerer Länge gerechnet werden. Das Resultat dabei ist eine drastische Reduzierung der Zahl der notwendigen Multiplikationen [1, 14, 20, 21]. Die Einzelheiten der schnellen Fouriertransformation sollen hier nicht dargestellt werden, an dieser Stelle interessiert lediglich ein möglicher Signalflußgraph und seine sequentielle bzw. parallele Realisierung auf programmierbaren Rechenwerken.

Es gibt eine Vielzahl unterschiedlicher Signalflußgraphen, von denen hier exemplarisch derjenige verwendet wird, der einen "Radix 2-in-place-decimation in time"-Algorithmus beschreibt (Bild (3.2-11)). Dabei wird der Übersichtlichkeit wegen die Transformationslänge auf $M = 8$ beschränkt.

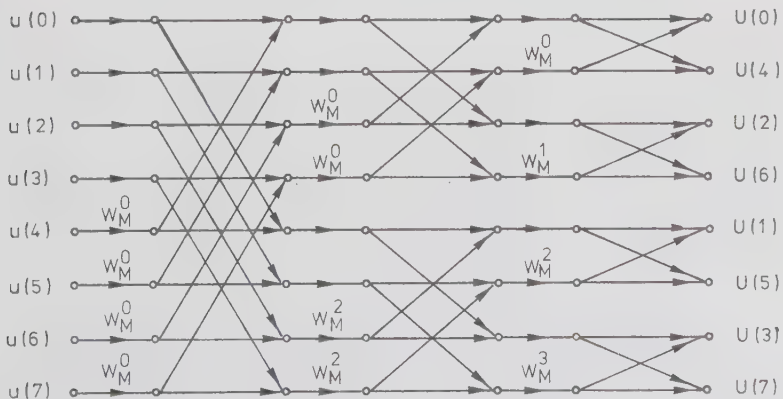


Bild (3.2-11) Signalflußgraph für "Bit-Reversal" am Ausgang
(entnommen aus [21])

Bild (3.2-11) zeigt, daß die Folge der Eingangswerte $u(k)$, $k = 0(1)7$, in 3 Stufen umgerechnet wird in eine Folge von Transformationswerten $U(\mu)$, $\mu = 0(1)7$.

Für den allgemeinen Fall einer FFT der Länge $M = 2^m$, $m \in \mathbb{N}_0$ ergibt sich die Zahl m der Stufen zu:

$$m = \text{ld } M \quad .$$

In jeder der zu berechnenden Stufen des Signalflußgraphen ist die Grundstruktur der sogenannte "Butterfly", (Bild (3.2-12)), der $M/2$ Mal gerechnet werden muß. Insgesamt erhält man damit für die Zahl N der "Butterflies":

$$N = \frac{M}{2} \text{ld } M \quad , \quad M = 2^m, m \in \mathbb{N}_0 \quad (3.2-32)$$

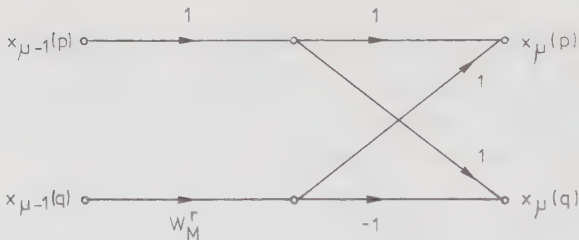


Bild (3.2-12) Radix 2-Butterflystruktur

Aus Bild (3.2-12) ergeben sich für die Variablen $x(p)$ und $x(q)$ folgende Zusammenhänge

$$\begin{aligned} x_{\mu}(p) &= x_{\mu-1}(p) + W_M^r x_{\mu-1}(q) \\ x_{\mu}(q) &= x_{\mu-1}(p) - W_M^r x_{\mu-1}(q) \end{aligned} \quad . \quad (3.2-33)$$

Berücksichtigt man, daß $x(p)$ und $x(q)$ sowie W_M^r komplexe Daten sind, so kann man Gl. (3.2-33) als reelle Matrixgleichung angeben [22]:

$$\begin{bmatrix} \operatorname{Re}\{x_{\mu}(p)\} \\ \operatorname{Im}\{x_{\mu}(p)\} \\ \operatorname{Re}\{x_{\mu}(q)\} \\ \operatorname{Im}\{x_{\mu}(q)\} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cos\alpha & \sin\alpha \\ 0 & 1 & -\sin\alpha & \cos\alpha \\ 1 & 0 & -\cos\alpha & -\sin\alpha \\ 0 & 1 & \sin\alpha & -\cos\alpha \end{bmatrix} \begin{bmatrix} \operatorname{Re}\{x_{\mu-1}(p)\} \\ \operatorname{Im}\{x_{\mu-1}(p)\} \\ \operatorname{Re}\{x_{\mu-1}(q)\} \\ \operatorname{Im}\{x_{\mu-1}(q)\} \end{bmatrix} \quad (3.2-34)$$

mit $\alpha = \frac{2\pi}{M} r$

bzw. $\mathbf{x}_{\mu} = \mathbf{A} \mathbf{x}_{\mu-1}$.

Die Berechnung von Gleichung (3.2-34) würde, wenn man die Matrix $\mathbf{A}$ mit dem Vektor $\mathbf{x}_{\mu-1}$ direkt multipliziert, durch die Bildung von 4 Produktsummen realisierbar sein:

$$x_{\mu v} = \sum_{\rho=1}^4 a_{\rho v} x_{(\mu-1)\rho} \quad v = 1, 2, 3, 4.$$

Bei Vernachlässigung der Multiplikationen mit dem Faktor Null sind dazu

12 Multiplikationen

und 8 Additionen

notwendig.

3.2.3.1 Realisierung eines FFT-Butterflies in einem Teilprozessor

Im Sinne der in den vorangegangenen Abschnitten verwendeten Matrizenbeschreibungen für kaskadierbare Teilprozessoren ist die in Gleichung (3.2-34) angegebene Matrix $\mathbf{A}$ bereits die Darstellung eines Teilprozessors mit 4 Ein- und Ausgangskanälen. Seine innere Struktur kann von einer direkten Realisierung der Produktsummen abgeleitet sein, wie dies in Bild (3.2-13) angegeben ist.

Ein Teilprozessor mit nur einem Multiplizierwerk ist in Bild (3.2-14) gezeigt.

Wegen der unterschiedlichen Zahl von Multiplikationen und Additionen läßt sich hier die in Bild (3.2-2) angegebene Struktur für Teilprozessoren nicht ökonomisch anwenden. Deshalb sind

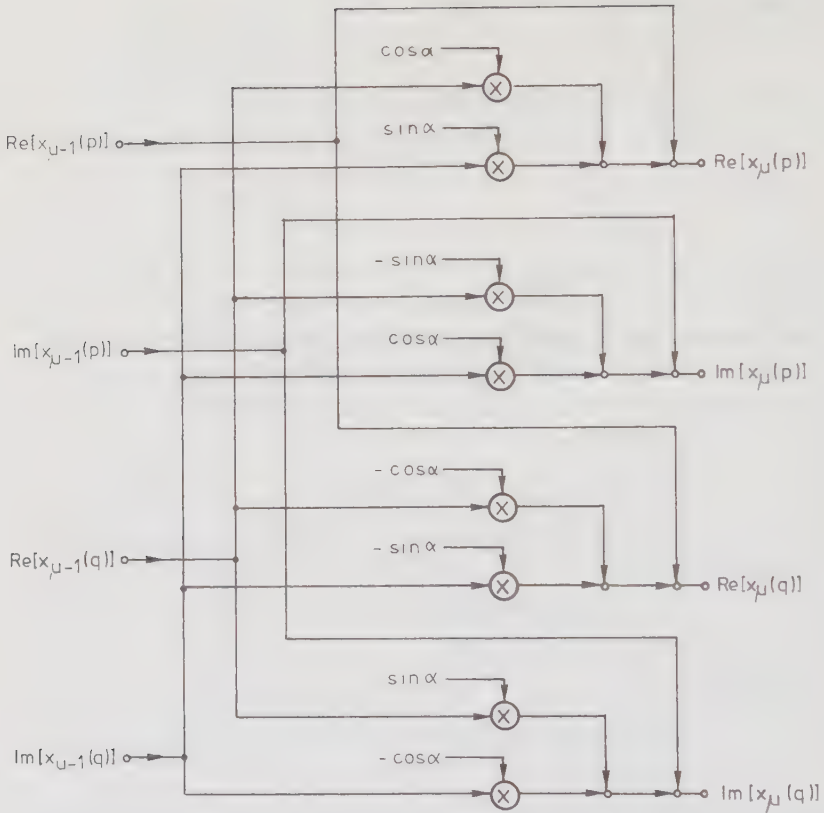
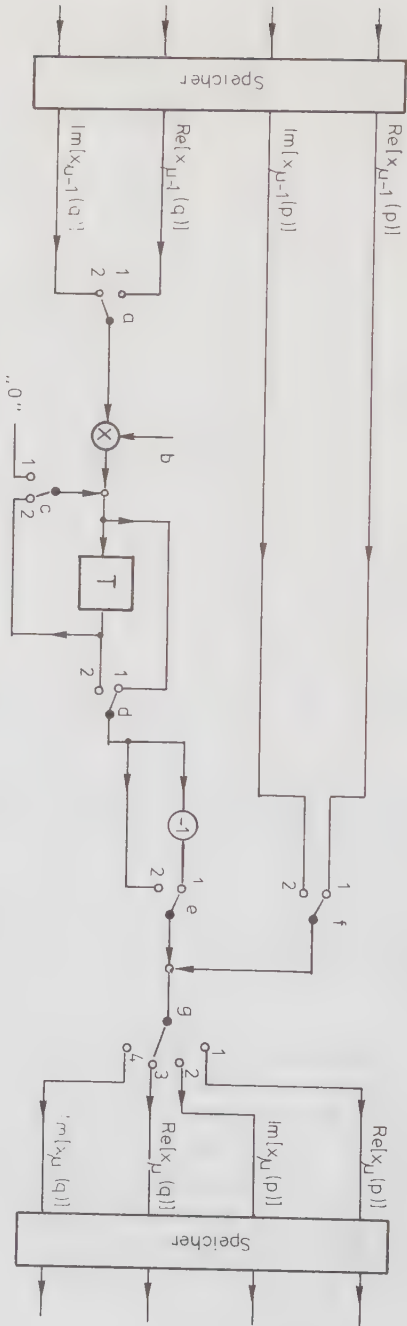


Bild (3.2-13) Direkte Realisierung der Produktsummenform in einem FFT-Teilprozessor

in Bild (3.2-14) ein Akkumulator, sowie auf der Ein- und Ausgangsseite des Teilprozessors Pufferspeicher zusätzlich angeordnet.

Einer dieser beiden Speicher lässt sich - bei der Realisierung der FFT durch ein Multiprozessorsystem - einem Teilprozessor fest zuordnen, wobei die Möglichkeit eines bilateralen Zugriffs direkt benachbarter Rechner vorgesehen werden muß.



Programmsequenz zur Berechnung eines Butterflies bei vollständiger Auslastung des Multiplizierers

Schritt	Mikrooperationen (Schalterstellungen, Daten)						
	a	b	c	d	e	f	g
1	1	cosa	1				
2	2	sina	2	1	1	1	3
3	1	cosa	1	2	2	1	1
4	2	-sina	2	1	1	2	4
(5)				2	2	2	2

Bild (3.2-15)

Sequentielle Realisierung eines FFT-Butterflies in dem Teilprozessor nach Bild (3.2-14)

Für einen kontinuierlichen Datenstrom am Eingang des Systems bedeutet dies, daß ein Butterfly in 4 Zyklen berechnet werden kann, wobei in jedem Zyklus multipliziert, und somit der Vorteil der FFT-Struktur vollständig ausgenutzt wird.

3.2.3.2 Berechnung der Schnellen Fourier Transformation durch Teilprozessoren

Im vorangegangenen Punkt 3.3.3.1 wurde die Realisierung eines Butterflies in einem Teilprozessor untersucht. Dabei beschreibt Gleichung (3.2-34) sein algorithmisches Verhalten.

Zur Berechnung der FFT werden nun drei Methoden mit unterschiedlichem Grad an Parallelarbeit angegeben.

a) Verwendung eines Teilprozessors

Werden in Bild (3.2-14,-15) die Ausgänge des Teilprozessors mit den Eingängen verbunden, so läßt sich der FFT-Algorithmus offensichtlich direkt realisieren, wenn die Programmsteuerung die entsprechende Sequenz von Adressen für den Speicher liefert (Bild (3.2-16)). Für den in Bild (3.2-14) angegebenen Teilprozessor bedeutet dies, daß der Ein- und Ausgangsspeicher eine Einheit darstellt.

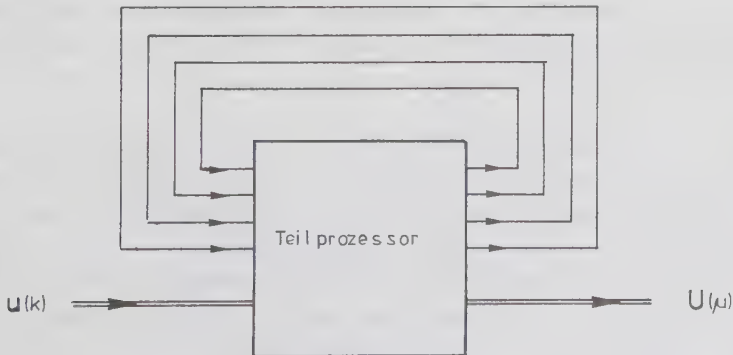


Bild (3.2-16) Rückgekoppelte Anordnung eines Teilprozessors zur Realisierung der FFT ohne Parallelarbeit

b) Verwendung von m Teilprozessoren

Eine weit höhere Rechengeschwindigkeit kann erreicht werden durch die Verwendung von so vielen Teilprozessoren, wie die zu berechnende FFT Spalten hat, also z.B. 10 Prozessoren für $M = 1024$.

Für die Beschaltung der Rechenwerke wird in [20] eine Struktur vorgeschlagen, die hier zitiert werden soll. Dabei beschränkt sich die Darstellung wieder auf eine Transformationslänge von $M = 8$, es werden also 3 Teilprozessoren eingesetzt, Bild (3.2-17). Es wird hierbei der Algorithmus von Bild (3.2-11) realisiert.

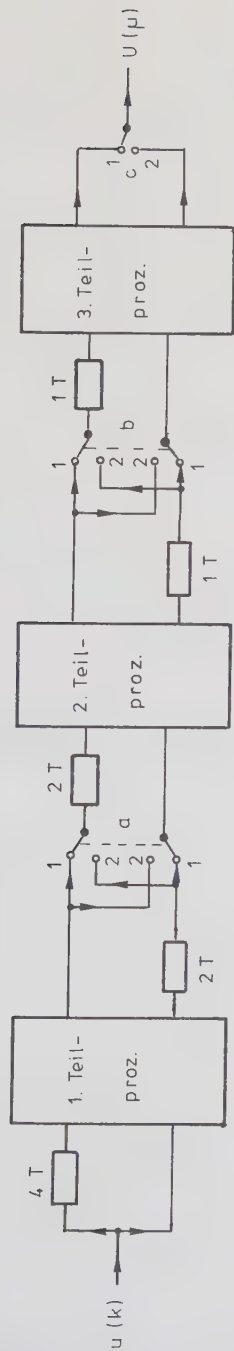
Bild (3.2-17) zeigt, daß die Eingangsfolge $u(k)$ mit dem ersten Teilprozessor direkt sowie über einen Pufferspeicher der Länge $4 = \frac{M}{2}$ verbunden ist. Dem entspricht in Bild (3.2-11) die Tatsache, daß in der ersten Stufe die Indizes k der beiden Eingangsvariablen für jeden Butterfly die Differenz $M/2$ haben.

Entsprechendes gilt für die Indexdifferenzen der folgenden Stufen. Sie sind $M/4$, $M/8$, ... 1.

Zusätzlich zu den Pufferspeichern sind in Bild (3.2-17) noch zwei Umschalter vorgesehen, die die in Bild (3.2-11) vorgegebenen Datenpfade realisieren.

Diese zuletzt erwähnten Schalter haben keinen Einfluß auf die innere Struktur der Teilprozessoren, denn sie sind realisierbar durch vom Programm vorgegebene feste Adressen, unter denen die Daten vom Eingangspufferspeicher ausgelesen werden.

Die in Bild(3.2-17) gezeigte Kaskade von Teilprozessoren wird insbesondere dann effektiv genutzt, wenn die Folge $u(k)$ wesentlich länger als die Transformationslänge M ist, und wenn von den einlaufenden Daten ständig blockweise die FFT gerechnet werden soll. Dann nämlich muß der erste Teilprozessor nicht warten, bis der Pufferspeicher der Länge $M/2$ gefüllt ist, sondern kann währenddessen ein anderes Segment der Folge $u(k)$ berechnen.



Schalterstellungen	Schalter	Stellung	für k
a		1	0, 1, 4, 5, 8, 9, ...
		2	2, 3, 6, 7, 10, 11, ...
b		1	gerade
		2	ungerade
c		1	gerade
		2	ungerade

Bild (3.2-17) Kaskadierung von Teilprozessoren zur Berechnung der FFT der Länge $M=8$

Wird nur ein Datenblock transformiert, kann während dem Einlaufen der ersten $M/2$ Datenwerte nicht gerechnet werden, das System wird dann nicht vollständig ausgenutzt.

c) Verwendung von $(M/2) \cdot m$ Teilprozessoren

Eine außerordentlich hohe Geschwindigkeit bei der Berechnung der FFT erreicht man durch die Verwendung je eines Prozessors pro Butterfly. Dies führt bereits bei kurzen Transformationslängen zu unrealistischen Konfigurationen, sie sind hier der Vollständigkeit halber angeführt.

Es ist auch dabei sinnvoll, mit je $M/2$ Prozessoren eine Spalte der FFT zu berechnen und das Ergebnis parallel an die nächste Stufe weiterzuleiten.

Weil für einen Ausgangswert $U(\mu)$ der FFT sämtliche Spaltenzwischenenergebnisse notwendig sind, muß jede "Prozessorspalte" (Bild 3.2-18) auf die Ergebnisse ihrer Vorgängerin warten. Dies bedeutet für die Rechenzeit T_R , daß gilt:

$$T_R = m \cdot T_B$$

mit T_B = Rechenzeit für einen Butterfly in einem Teilprozessor, und für die Datendurchsatzrate R

$$R = \frac{1}{T_B} \quad ,$$

wenn jede Prozessorspalte ständig mit neu einlaufenden Eingangsdatenblöcken beschäftigt wird, wenn also die Bearbeitung einer Folge von Datenblöcken und nicht die eines einzelnen Blockes realisiert werden soll.

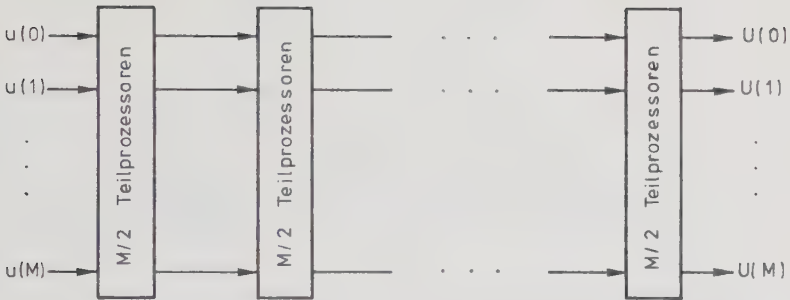


Bild (3.2-18) Anordnung von $\frac{M}{2}$ ld M Teilprozessoren zur vollständigen Parallelisierung der FFT

3.2.4 Potenzreihenentwicklung

Ein häufig angewandtes Verfahren zur Berechnung transzender Funktionen ist ihre Entwicklung in eine Potenzreihe [23]:

$$f(x) = \sum_{\mu=0}^n a_{\mu} x^{\mu} \quad , \quad n \text{ entsprechend der geforderten Genauigkeit.} \quad (3.2-35)$$

In Gleichung (3.2-35) ist dabei neben der Produktsummenbildung noch die Berechnung der Potenzen von x zu realisieren. Die Darstellung von Gleichung (3.2-35) nach dem Hornerschema liefert:

$$f(x) = (\dots ((a_n x + a_{n-1})x + a_{n-2})x + \dots)x + a_0 \quad (3.2-36)$$

Daraus kann unmittelbar eine realisierende Struktur für die Kaskadenschaltung von Teilprozessoren angegeben werden, Bild (3.2-19).

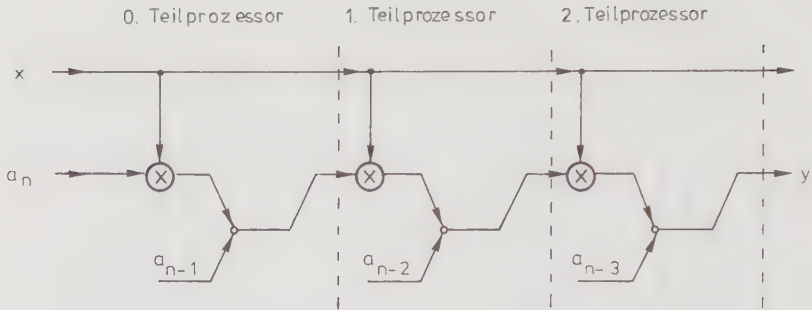


Bild (3.2-19) Realisierung des Hornerschemas mit Teilprozessoren

Entsprechend dem Vorgehen bei nichtrekursiven digitalen Filtern im Abschnitt 3.2.1 kann auch hier eine Matrix $\mathbf{A}$ angegeben werden, die allerdings jetzt von der Variablen x abhängt

$$\mathbf{A}^{(\mu)}(x) = \begin{bmatrix} 1 & 0 \\ a_{n-1-\mu} & x \end{bmatrix} \quad (3.2-37)$$

Mit Gleichung (3.2-13) wird aus (3.2-37)

$$\prod_{\mu=n-1}^0 \mathbf{A}^{(\mu)}(x) = \begin{bmatrix} 1 & 0 \\ \sum_{\mu=0}^{n-1} x^{n-1-\mu} a_{n-1-\mu} & x^n \end{bmatrix}$$

Für den Eingangsvektor $(1, a_n)^T$ ergibt sich somit am Ausgang y des $(n-1)$ -ten Teilprozessors

$$\begin{bmatrix} 1 \\ y \end{bmatrix} = \begin{bmatrix} n-1 \\ \prod_{\mu=0} \mathbf{A}^{(\mu)}(x) \end{bmatrix} \cdot \begin{bmatrix} 1 \\ a_n \end{bmatrix} = \begin{bmatrix} 1 \\ \sum_{\mu=0}^n x^{\mu} a_{\mu} \end{bmatrix} \quad \text{mit}$$

a_n = Eingangsvariable des Systems

$y^{(n)}$ = Ausgangsvariable des Systems.

Die Zahl der Multiplikationen beträgt bei diesem Verfahren n .

Würde man Gleichung (3.2-35) direkt mit vorab berechneten Potenzen von x realisieren, so wären

$$2n-1$$

Multiplikationen erforderlich.

3.2.5 Nichtrekursive zweidimensionale Filterung vom Blockgrad 0

Zweidimensionale digitale Filtertechniken werden hauptsächlich in der Bildverarbeitung angewendet. Dabei wird ein endliches zweidimensionales Datenfeld umgerechnet nach der Beziehung

$$y(\ell_1, \ell_2) = \sum_{\lambda_1=0}^{\ell_1} \sum_{\lambda_2=0}^{\ell_2} u(\lambda_1, \lambda_2) h_0(\ell_1 - \lambda_1, \ell_2 - \lambda_2) \quad (3.2-38)$$

wobei $u(\ell_1, \ell_2)$ das zu filternde Datenfeld in den beiden Dimensionen ℓ_1 und ℓ_2 , und $h_0(\ell_1, \ell_2)$ die Impulsantwort des zweidimensionalen Filters ist. Sie ist im nichtrekursiven Fall in beiden Dimensionen endlich.

Gleichung (3.2-38) ist eine zweidimensionale Faltungsbeziehung, aus ihr geht hervor, daß das Datenfeld $y(\ell_1, \ell_2)$ größer als das ursprüngliche Feld $u(\ell_1, \ell_2)$ ist. Hat nämlich $u(\ell_1, \ell_2)$ z.B. in der Richtung von ℓ_1 $L_1(u)$, und $h_0(\ell_1, \ell_2)$ $L_1(h_0)$ Werte, so hat $y(\ell_1, \ell_2)$ insgesamt $L_1(u) + L_1(h_0) - 1$ Werte in ℓ_1 Richtung, für die Richtung ℓ_2 ergibt sich entsprechend $L_2(u) + L_2(h_0) - 1$, so daß zu der ursprünglichen Feldfläche $L_1(u) \cdot L_2(u)$ nach der Filterung noch

$$\Delta F = (L_1(h_0) - 1)L_2(u) + (L_2(h_0) - 1)(L_1(u) + L_1(h_0) - 1)$$

Feldelemente hinzukommen. Bei einer ursprünglichen Feldausdehnung von z.B. $L_1(u) = L_2(u) = 256$ und mit $L_1(h_0) = L_2(h_0) = 3$ ergibt sich somit ein Flächenzuwachs um 1028 Punkte. Dies

entspricht ca. 1,6%. Bei einer mehrmaligen Filterung würde somit das Bild ständig in seinen Ausmaßen zunehmen. Diesen Nachteil vermeidet die Methode, die in [24] angegeben ist. Sie soll im folgenden als Beispiel zur zweidimensionalen Signalverarbeitung näher betrachtet werden.

3.2.5.1 Verarbeitungsprinzip

Der in [24] angegebene Algorithmus wird dargestellt als

$$\mathbf{v} = \mathbf{G} \cdot \mathbf{u} \quad , \quad (3.2-39)$$

wobei $\mathbf{G}$ das System beschreibt. $\mathbf{v}$ und $\mathbf{u}$ sind vektorielle Beschreibungen 2-dimensionalen Datenfelder, Bild (3.2-20) zeigt ihre Entstehung ^{*)}: das Feld $u(l_1, l_2)$ wird zum Vektor $\mathbf{u}$, indem aufeinanderfolgende Zeilen als Teilvektoren interpretiert und diese dann zu einem Gesamtvektor $\mathbf{u}$ zusammengefügt werden:

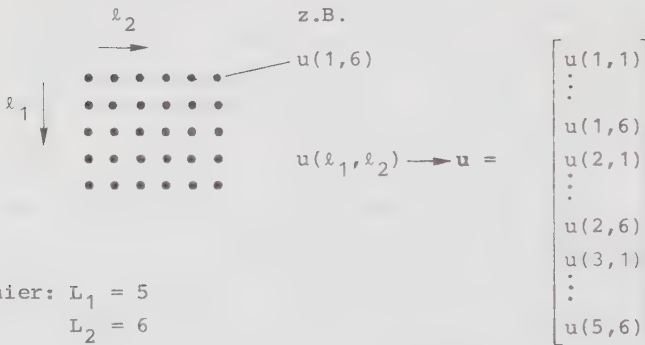


Bild (3.2-20) Darstellung eines zweidimensionalen Datenfeldes der Größe $L_1 \times L_2$ durch einen $L_1 L_2$ -dimensionalen Vektor

^{*)}

Bild (3.2-20, 21, 22) sind entnommen aus [24]

Es sei nur am Rande bemerkt, daß diese Darstellung eines zweidimensionalen Datenfeldes der Bildübertragung beim Fernsehen entspricht- auch dabei wird das Bild zeilenweise abgetastet und nach der Übertragung auf die gleiche Art wieder zusammengesetzt.

Bild (3.2-21) zeigt an einem Beispiel, wie ein Ergebnisbildpunkt $v(l_1, l_2)$ im Fall einer nichtrekursiven 2-dimensionalen Filterung errechnet wird. Die Filterkoeffizienten, die durch "x", "+", "0" angedeutet sind, werden mit dem Punkt $u(l_1, l_2)$, sowie mit seinen direkten Nachbarn multipliziert, und die Produkte aufsummiert. Dem entspricht Gleichung (3.2-38) im Innern des Feldes.

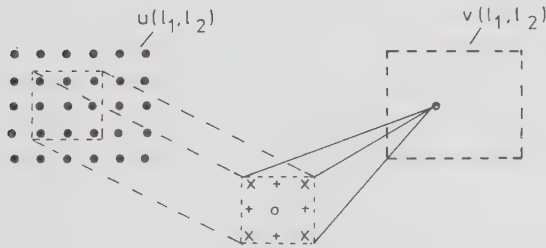


Bild (3.2-21) Errechnen von $v(l_1, l_2)$ aus $u(l_1, l_2)$ und seinen direkten Nachbarn. Der Koeffizientensatz ist hier zirkularsymmetrisch bezüglich des Koeffizienten "0"

Aus Bild (3.2-22) geht direkt hervor, daß der Ergebnisvektor $\mathbf{v}$ die gleiche Dimension wie der Vektor $\mathbf{u}$ hat, so daß sich durch die Feldverarbeitung keine Vergrößerung des Datenfeldes ergibt.

$\underline{\mathbf{G}}$ ist dabei eine $L_1 L_2 \times L_1 L_2$ Matrix, was z.B. bei einer Kantenlänge von $L_1 = L_2 = 512$ immerhin zu 2^{36} Matrixelementen führt. Allerdings ist $\underline{\mathbf{G}}$ nur schwach besetzt und hat zudem in den Diagonalen nur wenige unterschiedliche Werte, so daß die gesamte Matrix $\underline{\mathbf{G}}$ im hier angegebenen Beispiel mit 3 voneinander verschiedenen Koeffizienten beschreibbar ist. Sie muß daher nicht vollständig abgespeichert werden.

3.2.5.2 Modifikation des Algorithmus

Bild (3.2-22) zeigt, daß zur Berechnung eines Ergebniswertes $v(l_1, l_2)$ entweder 4, 6 oder 9 Multiplikationen bzw. 3, 5 oder 8 Additionen erforderlich sind. D.h., der Algorithmus zur Berechnung von $\underline{\mathbf{G}} \cdot \mathbf{u}$ muß in Abhängigkeit von l_1 und l_2 variiert werden. Dazu ist ein vergleichsweise hoher Rechen- und damit Zeitaufwand erforderlich, so daß eine Modifikation des Algorithmus wünschenswert erscheint, die diesen Nachteil vermeidet. Hierzu werden im folgenden drei Methoden angegeben.

a) Realisierung durch ein eindimensionales zeitinvariantes nichtrekursives Filter

Es wird die Matrix $\underline{\mathbf{G}}$ erweitert derart, daß sich in sämtlichen Diagonalen gleich viele Elemente befinden, die zudem innerhalb jeder Diagonalen gleich sind. Es entsteht so die Matrix $\underline{\mathbf{G}}'$. Sie wird multipliziert mit einem modifizierten Bildvektor $\mathbf{u}'$ um den - wiederum modifizierten - Ergebnisvektor $\mathbf{v}'$ zu erhalten. Bild (3.2-23) zeigt dies für das Beispiel in Bild (3.2-22).

Der Vektor $\mathbf{u}'$ geht dabei aus dem Vektor $\mathbf{u}$ durch Erweiterung mit Nullelementen hervor. Der Ergebnisvektor $\mathbf{v}'$ unterscheidet sich von $\mathbf{v}$ lediglich dadurch, daß die Elemente $\mathbf{v}'(l_1, 7)$ mit $l_1 \in (1, 2, 3, 4, 5)$ hinzukommen. Durch einfaches Streichen dieser Elemente wird also aus $\mathbf{v}'$ der gesuchte Vektor $\mathbf{v}$.

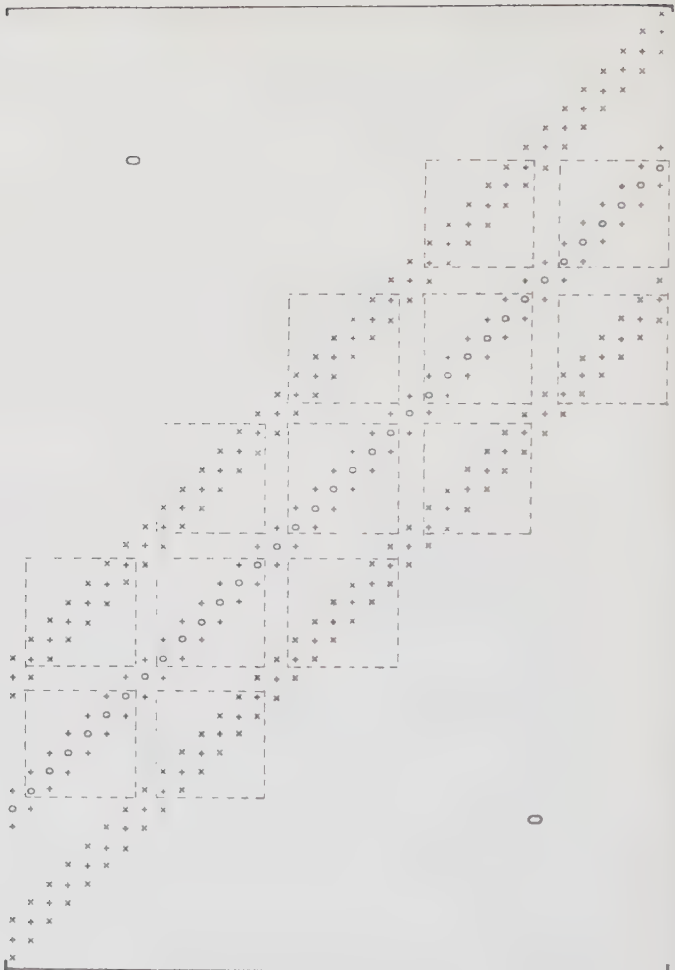


Bild (3.2 - 23)

Detaillierte Darstellung des Ausdrucks $G' u' = v'$

0	$v'(1, 1)$
0	•
0	•
0	$v'(1, 7)$
0	$v'(2, 1)$
0	•
0	•
0	•
0	$v'(2, 7)$
0	$v'(3, 1)$
0	•
0	•
0	•
0	$v'(3, 7)$
0	$v'(4, 1)$
0	•
0	•
0	•
0	$v'(4, 7)$
0	$v'(5, 1)$
0	•
0	•
0	$v'(5, 7)$

Dieses Erweiterungsprinzip wird nun für das in Bild (3.2-21) angegebene Koeffizientenfeld für beliebige Feldausdehnungen L_1 und L_2 beschrieben.

Der Bildvektor $\mathbf{u}$ in Bild (3.2-22) hat die Dimension $L_1 L_2$. Daraus resultiert für die Matrix $\underline{\mathbf{G}}$ die Dimension $L_1 L_2 \times L_1 L_2$ und die erweiterte Matrix $\underline{\mathbf{G}}'$ wird zu einer $[L_1(L_2+1) \times ((L_1+2)(L_2+1)+2)]$ -Matrix, wenn zwischen jeden besetzten Diagonalblock noch eine Nulldiagonale eingefügt wird.

Ziel ist, den Ausdruck

$$\mathbf{u} = \underline{\mathbf{G}} \cdot \mathbf{v}$$

mittels der Matrix $\underline{\mathbf{G}}'$ zu errechnen. Dazu wird folgender Ansatz gemacht:

$$\mathbf{u} = \mathbf{M}_2' \underline{\mathbf{G}}' \mathbf{M}_1' \cdot \mathbf{v} \quad (3.2-40)$$

also

$$\underline{\mathbf{G}} = \mathbf{M}_2' \underline{\mathbf{G}}' \mathbf{M}_1' \quad . \quad (3.2-41)$$

Da für $\underline{\mathbf{G}}'$ eine $[L_1(L_2+1) \times ((L_1+2)(L_2+1)+2)]$ -Matrix gewählt wurde, folgt aus Gl. (3.2-41), daß

$\mathbf{M}_1'$ eine $[((L_1+2)(L_2+1)+2) \times (L_1 L_2)]$ -Matrix

und

$\mathbf{M}_2'$ eine $[(L_1 L_2) \times (L_1(L_2+1))]$ -Matrix

sein muß.

Weiterhin sollen die Elemente von $\mathbf{M}_1'$ und $\mathbf{M}_2'$ entweder Null oder Eins sein, damit keine nichttrivialen Multiplikationen eingeführt werden. Es wird deshalb gewählt:

für $\mathbf{M}_1'$:

$$\mathbf{M}_1' = \left[\begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \right] \left\{ \begin{array}{l} \text{---} L_2+2 \text{ Nullzeilen} \\ \text{---} 1 \text{ Nullzeile} \\ \text{---} 1 \text{ Nullzeile} \\ \text{---} 1 \text{ Nullzeile} \\ \text{---} L_2+3 \text{ Nullzeilen} \end{array} \right. \left. \begin{array}{l} L_1 \text{ Einheits-} \\ \text{matrizen } \mathbf{E} \\ \text{der Dimen-} \\ \text{sion } L_2 \end{array} \right\}$$

$\mathbf{M}_1' = \left[\begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \right] \left\{ \begin{array}{l} \text{---} L_2+2 \text{ Nullzeilen} \\ \text{---} 1 \text{ Nullzeile} \\ \text{---} 1 \text{ Nullzeile} \\ \text{---} 1 \text{ Nullzeile} \\ \text{---} L_2+3 \text{ Nullzeilen} \end{array} \right. \left. \begin{array}{l} L_1 \text{ Einheits-} \\ \text{matrizen } \mathbf{E} \\ \text{der Dimen-} \\ \text{sion } L_2 \end{array} \right\}$

mit $L_1 L_2$ Spalten

(3.2-42)

Bei dem Produkt $\mathbf{M}'_2(\underline{\mathbf{G}}'\mathbf{M}'_1)$ eliminiert $\mathbf{M}'_2$ in $\underline{\mathbf{G}}'\mathbf{M}'_1$ die (n-7)-te Zeile mit $n = 1, 2, 3, 4$ und schiebt die Zeilen 8 ... 13 zu den Zeilennummern 7 ... 12, die Zeilen 15 ... 20 nach 13 ... 18 usw., so daß $\mathbf{M}'_2\underline{\mathbf{G}}'\mathbf{M}'_1 = \underline{\mathbf{G}}$ wird.

Der Vorteil bei der Berechnung von

$$\mathbf{v} = \mathbf{M}'_2\underline{\mathbf{G}}'\mathbf{M}'_1\mathbf{u}$$

gegenüber $\mathbf{v} = \underline{\mathbf{G}}\mathbf{u}$

liegt nun darin, daß einerseits die Ausdrücke

$$\mathbf{u}' = \mathbf{M}'_1\mathbf{u} \quad (3.2-44)$$

und

$$\mathbf{v} = \mathbf{M}'_2\mathbf{v}' \quad (3.2-45)$$

mit

$$\mathbf{v}' = \underline{\mathbf{G}}'\mathbf{u}' \quad (3.2-46)$$

sehr einfach realisierbar sind wie in Abschnitt d) gezeigt wird, und andererseits das Produkt aus Gleichung (3.2-46) durch einen Produktsummenausdruck darstellbar ist. Dies zeigt das Folgende.

Werden die Vektoren $\mathbf{u}'$ und $\mathbf{v}'$ durch ihre Elemente $u'(k)$, $k = 1 \dots [(L_1+2)(L_2+1)+2]$ und $v'(k)$, $k = 1 \dots [L_1(L_2+1)]$ dargestellt, ergibt sich für das Produkt $\underline{\mathbf{G}}'\mathbf{u}'$

$$v'(k) = \sum_{\mu=1}^{(L_1+2)(L_2+1)+2} g'(k, \mu) u'(\mu) \quad (3.2-47)$$

wenn $g'(k, \mu)$ ein Element der Matrix $\underline{\mathbf{G}}'$ in der k-ten Zeile und μ -ten Spalte ist.

Zwei Eigenschaften der Matrix $\underline{\mathbf{G}}'$ gestatten nun die Vereinfachung von Gleichung (3.2-47):

1. Da nur die Diagonalen besetzt sind, gilt $g'(k, \mu) \equiv 0$ für $\mu < k$ und $\mu > k+n$, wenn $n = (\text{Zahl der Spalten von } \underline{\mathbf{G}}') - (\text{Zahl der Zeilen von } \underline{\mathbf{G}}')$ ist (hier $n=2L_2+4$).

Damit beschränkt sich der Summenausdruck in (3.2-47) auf

$$v'(k) = \sum_{\mu=k}^{k+n} g'(k, \mu) u'(\mu) \quad . \quad (3.2-48)$$

2. Da alle Elemente in jeder Diagonale identisch sind, gilt $g'(k, \mu) = g'(k+\lambda, \mu+\lambda)$ für $k+\lambda \leq L_1(L_2+1)$ und $\mu+\lambda \leq [(L_1+2)(L_2+1)+2]$. Substituiert man in (3.2-48) gemäß

$$\mu: = \mu + k ,$$

ergibt sich

$$v'(k) = \sum_{\mu=0}^n g'(k, \mu+k) u'(\mu+k)$$

und man erhält mit der angegebenen Eigenschaft für $g'(k, \mu)$

$$v'(k) = \sum_{\mu=0}^n g'(\mu) u'(\mu+k) \quad (3.2-49)$$

für $k = 1 \dots [L_1(L_2+1)]$

wenn hier $g'(\mu)$ für $g'(1, \mu+1)$, also für die Koeffizientenfolge der ersten Zeile der Matrix $\underline{G}'$ steht.

Damit wird die Berechnung von Gleichung (3.2-46) durch Gleichung (3.2-49) auf einen eindimensionalen, nicht-rekursiven und in k invarianten Filteralgorithmus zurückgeführt. $g'(\mu)$ ist dann die Impulsantwort des Systems der Länge $n+1$, und es läßt sich die in Bild (3.2-24) dargestellte realisierende Struktur angeben. Zur Errechnung des Vektors $\underline{v}'$ sind dabei $L_1(L_2+1)$ Filterzyklen erforderlich.

In Bild (3.2-24) realisieren die Systeme S_1' und S_2' die Operationen $\underline{M}_1' \underline{u}$ bzw. $\underline{M}_2' \underline{v}'$, indem sie in die Wertefolge $u(k)$ an den entsprechenden Stellen Nullen einfügen, bzw.

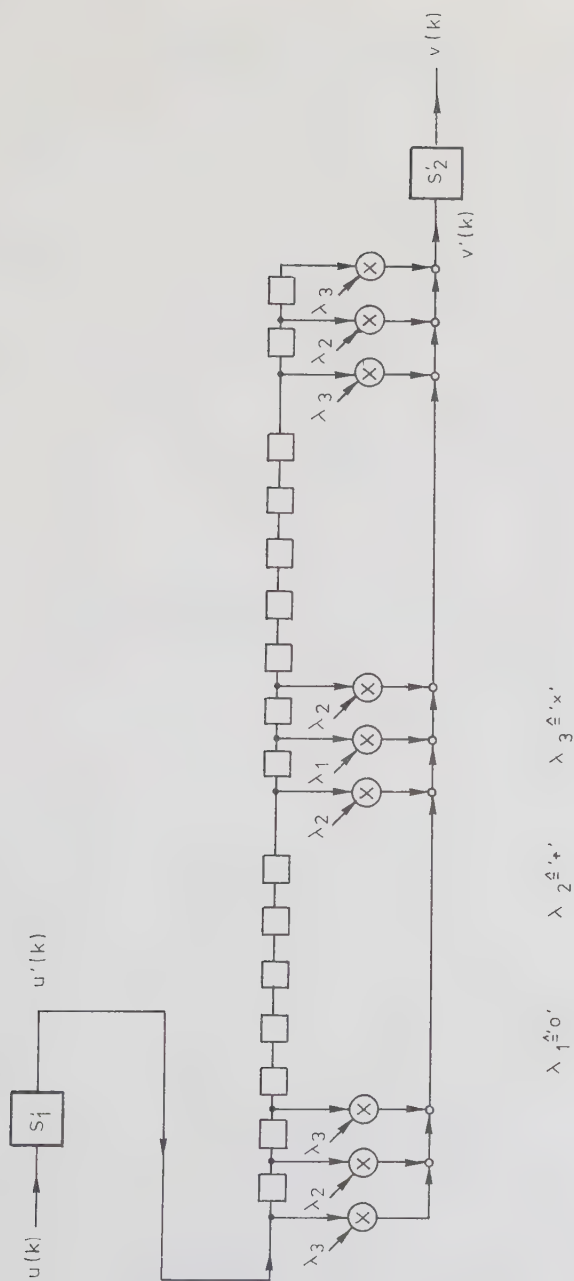


Bild (3.2-24)

Nichtrekursives eindimensionales Filter zur Realisierung der 2-dimensionalen nichtrekursiven Filterung vom Blockgrad Null nach Bild (3.2-21). Die Systeme S'_1 und S'_2 führen die Matrixoperationen $\mathbf{M}'_1 \mathbf{u}$ und $\mathbf{M}'_2 \mathbf{v}'$ aus.

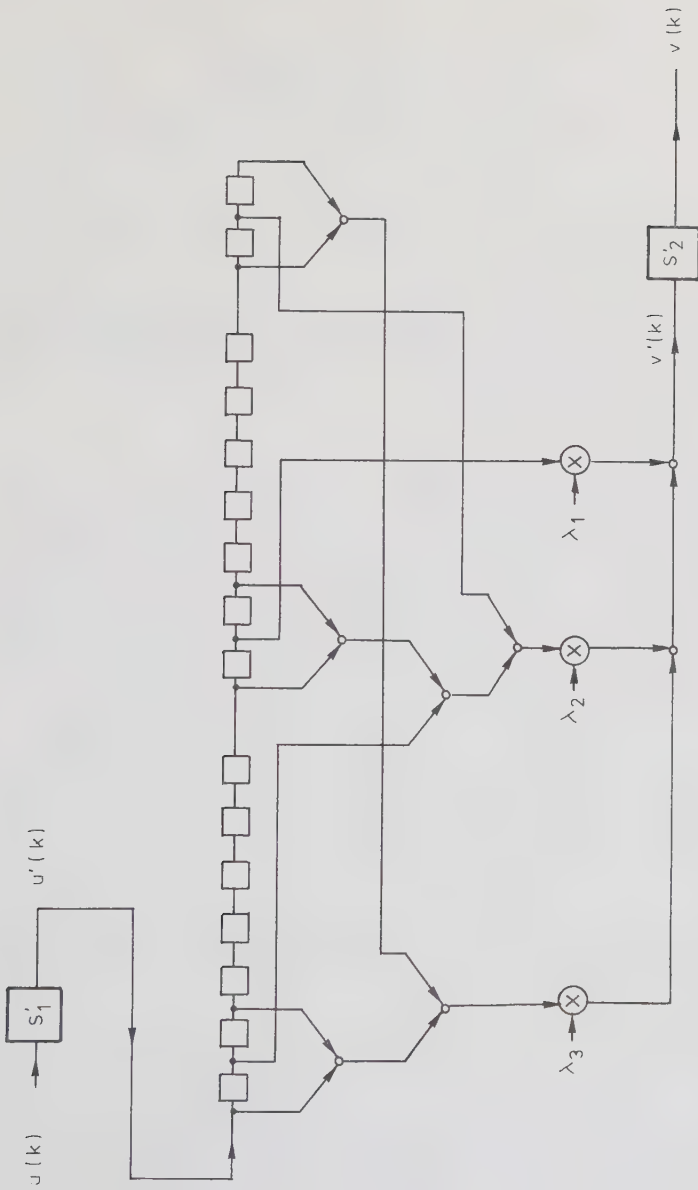


Bild (3.2-25), Struktur zur Realisierung des Algorithmus nach Gleichung (3.2-52)

bzw. in vektorieller Darstellung der Produktsumme

$$\mathbf{v}' = (\lambda_1, \lambda_2, \lambda_3) (\underline{\mathbf{g}}_1' \mathbf{u}, \underline{\mathbf{g}}_2' \mathbf{u}, \underline{\mathbf{g}}_3' \mathbf{u})^T. \quad (3.2-51)$$

Unter Verwendung von Gleichung (3.2-49) ergibt sich somit

$$\mathbf{v}'(k) = \sum_{v=1}^3 \lambda_v \sum_{\mu=0}^n g_v'(\mu) u'(k+\mu), \quad g_v' \in \{0, 1\}. \quad (3.2-52)$$

Die Bedeutung der Werte g_v' ist offensichtlich, es gelten für Gleichung (3.2-52) die gleichen Einschränkungen bzgl. des k wie für (3.2-49). Bild (3.2-25) zeigt eine realisierende Struktur.

c) Realisierung mit variablen Filterkoeffizienten

Die in Abschnitt a) und b) angewandte Methode zur Realisierung der Gleichung $\mathbf{v} = \underline{\mathbf{G}} \mathbf{u}$ diente primär dazu, den Algorithmus zur Berechnung des Produktes $\underline{\mathbf{G}} \mathbf{u}$ zu vereinfachen, indem die Eigenschaften des durch $\mathbf{u}$ repräsentierten Signals verändert wurden, um mit einem eindimensionalen nichtrekursiven Filter arbeiten zu können.

In diesem Abschnitt soll kurz ein Verfahren angegeben werden, das zwar einen höheren Speicheraufwand für die Darstellung der Matrix $\underline{\mathbf{G}}$ benötigt, dafür aber mit weniger Taktzyklen des Filters auskommt.

Bild (3.2-26) zeigt das Prinzip. Es wird eine modifizierte Matrix $\underline{\mathbf{G}}''$ verwendet, deren "innere Struktur" ähnlich der der Matrix $\underline{\mathbf{G}}$ ist, das bedeutet insbesondere, daß die Zahl der Elemente, die ungleich Null sind, nicht in allen Zeilen gleich ist.

Im folgenden wird diese Modifikation detaillierter beschrieben. Ziel ist, auch hier ein eindimensionales nichtrekursives Filter zur Realisierung des Algorithmus zu verwenden. Dazu werden entsprechend dem Vorgehen in Abschnitt a) modifizierte Vektoren und eine modifizierte Matrix eingeführt:

$$\mathbf{v}'' = \underline{\mathbf{G}}'' \mathbf{u}'' \quad . \quad (3.2-53)$$

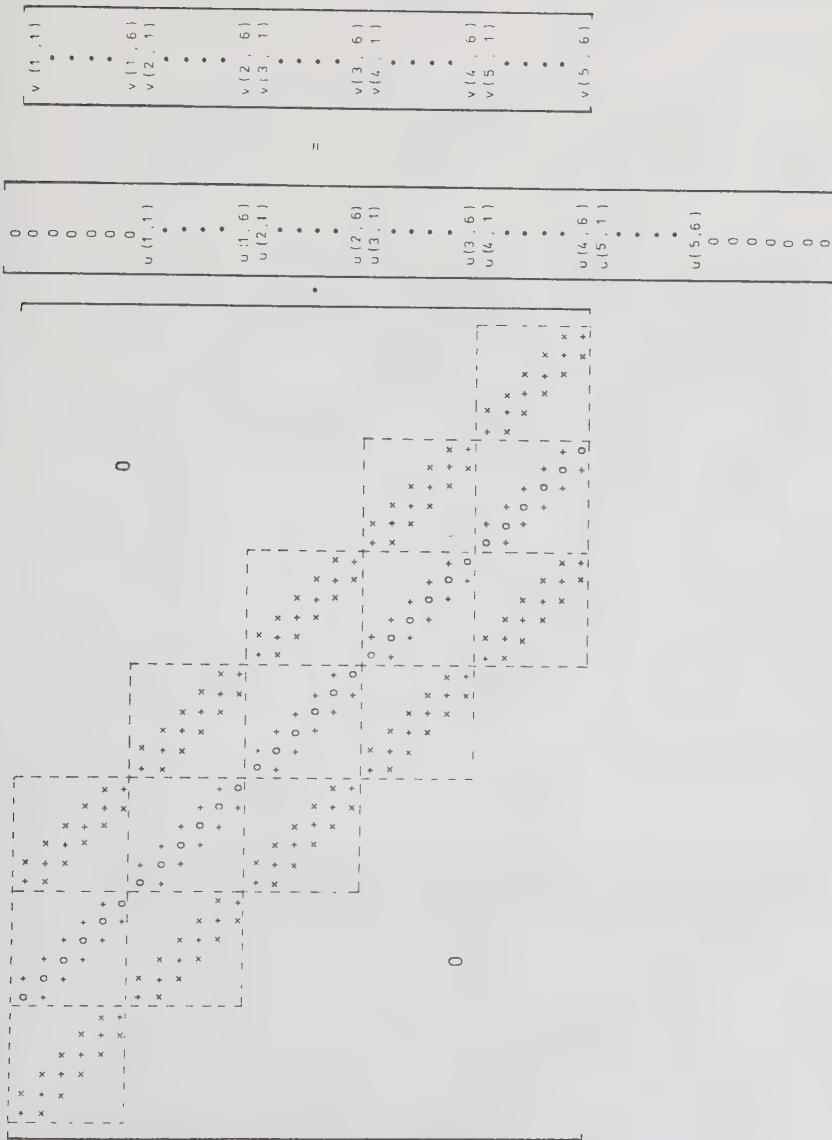


Bild (3.2.-26)

Detaillierte

Darstellung des Ausdrucks

$\bar{G}, U = V$

Es wird wieder der Ansatz

$$\underline{v} = \underline{M}_2'' \underline{G}'' \underline{M}_1'' \underline{u} \quad (3.2-54)$$

gemacht, mit

$$\underline{u}'' = \underline{M}_1'' \underline{u} \quad (3.2-55)$$

und

$$\underline{v} = \underline{M}_2'' \underline{v}'' \quad . \quad (3.2-56)$$

Um auch hier den Algorithmus auf ein eindimensionales nichtrekursives Filter mit konstanter Länge der Impulsantwort zurückführen zu können, wird die Matrix $\underline{G}''$ durch Erweiterung der Matrix $\underline{G}$ gebildet.

Die in Bild (3.2-22) angegebene Matrix $\underline{G}$ läßt sich auch als $L_1 \times L_1$ -Blockmatrix angeben, die aus $L_1 \times L_1$ Untermatrizen der Dimension $L_2 \times L_2$ besteht [24]:

$$\underline{G} = \begin{bmatrix} \underline{G}_0 & \underline{G}_1 & & 0 \\ \underline{G}_1 & \underline{G}_0 & \underline{G}_1 & \\ & \underline{G}_1 & \underline{G}_0 & \underline{G}_1 \\ & & \underline{G}_1 & \underline{G}_0 & \underline{G}_1 \\ 0 & & & \underline{G}_1 & \underline{G}_0 \end{bmatrix} \quad (3.2-57)$$

Die Matrizen $\underline{G}_0$ und $\underline{G}_1$ entsprechen den in Bild (3.2-22) gestrichelt eingezeichneten Blöcken der Matrix $\underline{G}$.

Die Matrix $\underline{G}''$ ergibt sich nun durch

$$\underline{G}'' = \begin{bmatrix} \underline{G}_1 & \underline{G}_0 & \underline{G}_1 & & \\ & \underline{G}_1 & \underline{G}_0 & \underline{G}_1 & 0 \\ & & \underline{G}_1 & \underline{G}_0 & \underline{G}_1 \\ & & & \underline{G}_1 & \underline{G}_0 & \underline{G}_1 \\ 0 & & & & \underline{G}_1 & \underline{G}_0 & \underline{G}_1 \end{bmatrix} \quad . \quad (3.2-58)$$

Aus Gleichung (3.2-39) und (3.2-54) gewinnt man eine Bestimmungsgleichung für die Matrizen $\mathbf{M}_1''$ und $\mathbf{M}_2''$:

$$\underline{\mathbf{G}} = \mathbf{M}_2'' \underline{\mathbf{G}}'' \mathbf{M}_1'' \quad (3.2-59)$$

Unter der Bedingung, daß die Elemente von $\mathbf{M}_1''$ und $\mathbf{M}_2''$ zur Vermeidung nichttrivialer Multiplikationen in (3.2-55 und -56) entweder Null oder Eins sein sollen, erfüllen

$$\mathbf{M}_1'' = \begin{bmatrix} \mathbf{O} & & & & \\ \mathbf{E} & & & & \\ & \mathbf{E} & & & \\ & & \mathbf{E} & & \\ & & & \mathbf{E} & \\ & & & & \mathbf{E} \\ & & & & & \mathbf{O} \end{bmatrix}$$

1 Nullzeile

} L_1 Einheitsmatrizen

1 Nullzeile

und

$$\mathbf{M}_2'' = \mathbf{E}$$

die Gleichung (3.2-59). Dabei haben die Nullmatrizen $\mathbf{O}$ sowie die Einheitsmatrizen $\mathbf{E}$ in $\mathbf{M}_1''$ die Dimension L_2 . $\mathbf{M}_2''$ ist eine $(L_1 L_2 \times L_1 L_2)$ -Einheitsmatrix.

Diese Beschreibung des Algorithmus mittels der Modifikationsmatrizen wird im Abschnitt 3.2.6 benötigt werden.

Entsprechend der Vorgehensweise der vorangegangenen beiden Abschnitte läßt sich auch hier der Vektor $\mathbf{u}''$ durch seine Elemente $u''(k)$, $k = 1 \dots [(L_1+2)L_2+2]$ angeben.

Die Multiplikation einer Zeile der Matrix $\underline{g}''$ mit dem Vektor $\underline{u}''$ wird dann durch den Ausdruck

$$v(k) = \sum_{\mu=0}^n g''(\mu, k) u''(k+\mu) \quad (3.2-60)$$

beschrieben. Dabei ist $n = 2L_2 + 2$ und $g''(\mu, k)$ variabel in k . Zur Verdeutlichung dieses letzten Punktes werden für das hier behandelte Beispiel einige Koeffizienten durch ihre Symbole explizit angegeben (ein fehlendes Koeffizientensymbol steht für $g''(\mu, k) = 0$):

$\mu =$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$k =$															
1		+	x					0	+					+	x
2	x	+	x				+	0	+				x	+	x
3	x	+	x				+	0	+				x	+	x
⋮															
6	x	+					+	0					x	+	
7		+	x					0	+					+	x
8	x	+	x				+	0	+				x	+	x
⋮															
29	x	+	x				+	0	+				x	+	x
30	x	+					+	0					x	+	

Hieraus wird deutlich, daß die Matrix $\underline{g}''$ durch nur drei unterschiedliche Koeffizientensätze darstellbar ist:

$$g''(\mu, k) = \begin{cases} g''(\mu, 1) & \text{für } k = 1 + nL_2, \quad n=0, (1), [L_1-1] \\ g''(\mu, L_2) & \text{für } k = nL_2, \quad n=1, (1), L_1 \\ g''(\mu, 2) & \text{sonst} \end{cases}$$

Ausgehend von Gleichung (3.2-60) läßt sich auch hier eine eindimensionale nichtrekursive Filterstruktur angeben, wobei die Impulsantwort abhängig von k ist. Dies zeigt Bild (3.2-27). Das System S_1 realisiert dabei Gleichung (3.2-55). Zur Berechnung des Vektors $\underline{v}$

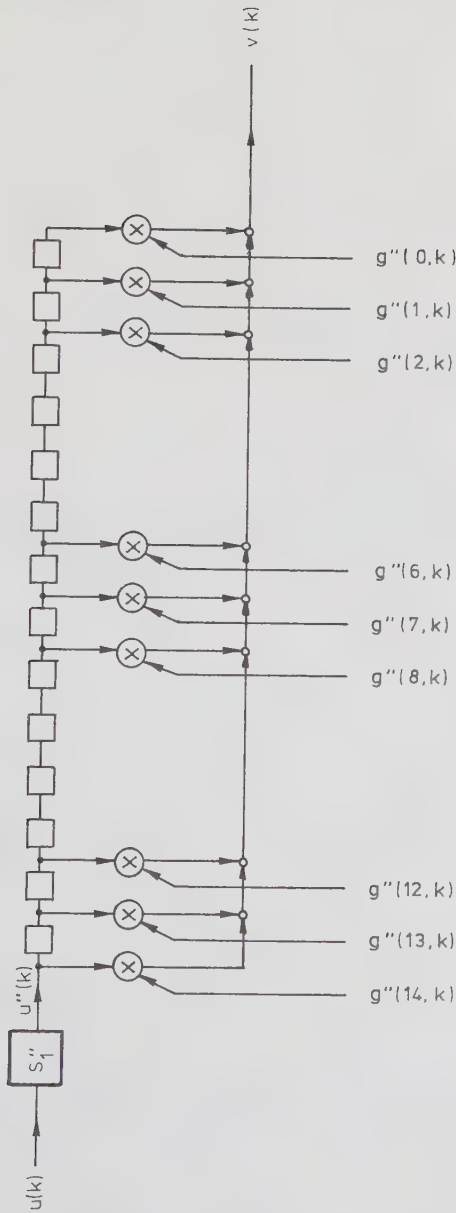


Bild. (3.2-27) Struktur des in k variablen, nichtrekursiven eindimensionalen Systems zur Realisierung von $\mathbf{v} = \mathbf{G} \mathbf{u}$

sind $L_1 L_2$ Filterzyklen notwendig. Es lassen sich auch hier die in Abschnitt 3.2.1 angegebenen Teilprozessoren einsetzen.

d) Realisierung der Modifikationsmatrizen

Die in den vorangegangenen Abschnitten eingeführten Matrizen M_1 und M_2 werden in den Bildern (3.2-24, -25, -27) symbolisch durch die Systeme S_1 und S_2 realisiert. Nachfolgend wird angegeben, wie sie in einem Signalverarbeitungssystem implementiert werden können.

Das System S_1' realisiert den Ausdruck

$$M_1' u$$

und bildet dabei den Vektor u' wie er im Abschnitt 3.2.5.2 a angegeben ist.

Wird u' als 2-dimensionales Datenfeld graphisch dargestellt, so kann man z.B. die in Bild (3.2-28) gezeigte Anordnung wählen.

$u'(1,1)$	0	0	0	0	0	0	0	
	0	u	u	u	u	u	u	z.B. $u'(2,7) = u(1,6)$
	0	u	u	u	u	u	u	u steht für
	0	u	u	u	u	u	u	ein Feldelement
	0	u	u	u	u	u	u	$u(l_1, l_2)$
	0	u	u	u	u	u	u	
	0	0	0	0	0	0	0	
	0	0						

Bild (3.2-28) Beispiel für eine 2-dimensionale Darstellung des Vektors u' als Feld $u'(l_1, l_2)$

u' wird also dadurch gebildet, daß das Feld $u(l_1, l_2)$ am oberen, linken und unteren Rand durch Nullelemente erweitert wird. Es müssen noch zwei weitere Elemente am unteren Rand hinzugefügt werden, um die Wirkung der Matrix M_1' zu realisieren.

Damit gilt für das Feld $u'(l_1, l_2)$:

$$u'(l_1, l_2) = \begin{cases} 0 & \text{für } (l_1=1 \text{ und/oder } l_2=1) \text{ oder } (l_1 > (L_1+1)) \\ u(l_1-1, l_2-1) & \text{sonst} \end{cases} \quad (3.2-61)$$

Die Verwirklichung dieses Datenfeldes kann entweder dadurch geschehen, daß in einem programmierbaren System die Elemente $u'(l_1, l_2) = 0$ gebildet werden durch Abfrage der Parameter l_1 und l_2 und bedingtes Nullsetzen der Systemeingangswerte, oder - und dies ist eine wesentlich einfachere Methode - durch Erweiterung des Speicherfeldes um 3 Zeilen sowie eine Spalte und durch Löschen der in (3.2-61) angegebenen Feldelemente.

Die Matrix M_1'' (Gl. (3.2-55)) erweitert den Vektor u in

$$u'' = M_1'' u$$

um $2(L_2+1)$ Nullelemente.

Bild (3.2-29) zeigt hierfür eine dem Bild (3.2-28) entsprechende Darstellung des Feldes $u''(l_1, l_2)$.

Dabei ist

$$u''(l_1, l_2) = \begin{cases} 0 & \text{für } (l_1 = 1, 2) \text{ oder } (l_1 = L_1+3, +4) \\ u(l_1-2, l_2) & \text{sonst} \end{cases} \quad (3.2-62)$$

die algorithmische Vorschrift zur Bildung des Produktes

$$u'' = M_1'' u \quad .$$

					0
0	0	0	0	0	0
u	u	u	u	u	u
u	u	u	u	u	u
u	u	u	u	u	u
u	u	u	u	u	u
u	u	u	u	u	u
u	u	u	u	u	u
0	0	0	0	0	0
0					

z.B. $u''(5,6) = u(3,6)$

Bild (3.2-29) 2-dimensionale Darstellung des Datenfeldes $u''(\ell_1, \ell_2)$

Auch hier bietet sich die Realisierung durch eine Erweiterung des Speicherbereiches um 4 Nullzeilen an.

Die Modifikationsmatrix M'_2 läßt sich dadurch verwirklichen, daß die Vektorelemente $v'(k)$ mit $k = n \cdot (L_2 + 1)$, $n = 1 \dots L_1$ unterdrückt werden, bzw. die Feldelemente $v'(\ell_1, L_2 + 1)$ für alle ℓ_1 zu Null gesetzt werden.

M''_2 muß, wie bereits in Abschnitt 3.6.4 erwähnt, nicht gesondert realisiert werden, da $M''_2 = E$.

3.2.5.3 Rechenaufwand

Für die vorgestellten Verfahren zur Berechnung des Ausdruckes $v = G u$ sind in der nachfolgenden Tabelle, die notwendigen Multiplikationen und Additionen für einige typische Fälle angegeben.

Algorithmus	Multiplikationen	normiert	Additionen	normiert
1. $\mathbf{v}=\mathbf{G} \cdot \mathbf{u}$ $L_1=L_2 = 64$ $= 128$ $= 256$ $= 512$	$9L_1L_2-6L_1-6L_2+4$ $36 \cdot 100$ $145 \cdot 924$ $586 \cdot 756$ $2 \cdot 353 \cdot 156$	 1 1 1 1	$8L_1L_2-6L_1-6L_2+4$ $32 \cdot 004$ $129 \cdot 540$ $521 \cdot 229$ $2 \cdot 091 \cdot 012$	 1 1 1 1
2. $\mathbf{v}'=\mathbf{G}' \cdot \mathbf{u}'$ $L_1=L_2 = 64$ $= 128$ $= 256$ $= 512$	$9L_1L_2+9L_1$ $37 \cdot 440$ $148 \cdot 608$ $592 \cdot 128$ $2 \cdot 363 \cdot 904$	 1.037 1.018 1.009 1.005	$8L_1L_2+8L_1$ $33 \cdot 280$ $132 \cdot 096$ $526 \cdot 336$ $2 \cdot 101 \cdot 248$	 1.040 1.020 1.010 1.005
3. $\mathbf{v}'=(\sum \mathbf{G}'_{\nu} \lambda_{\nu}) \mathbf{u}'$ $L_1=L_2 = 64$ $= 128$ $= 256$ $= 512$	$3L_1L_2+3L_1$ $12 \cdot 480$ $49 \cdot 536$ $197 \cdot 376$ $787 \cdot 968$	 0.346 0.339 0.336 0.335	$8L_1L_2+8L_1$ $33 \cdot 280$ $132 \cdot 096$ $526 \cdot 336$ $2 \cdot 101 \cdot 248$	 1.040 1.020 1.010 1.005
4. $\mathbf{v}''=\mathbf{G}'' \cdot \mathbf{u}''$ $L_1=L_2 = 64$ $= 128$ $= 256$ $= 512$	$9L_1L_2$ $36 \cdot 864$ $147 \cdot 456$ $589 \cdot 824$ $2 \cdot 359 \cdot 296$	 1.021 1.010 1.005 1.003	$8L_1L_2$ $32 \cdot 768$ $131 \cdot 072$ $524 \cdot 288$ $2 \cdot 097 \cdot 152$	 1.024 1.012 1.006 1.003

Tabelle : Vergleich des Rechenaufwandes (Es ist ein 3×3 Koeffizientenfeld zugrundegelegt)

Der Tabelle liegt das in diesem Abschnitt durchweg verwendete Beispiel eines Koeffizientenfeldes nach Bild (3.2-21) zugrunde. Als Parameter wird die Kantenlänge eines quadratischen Feldes $u(l_1, l_2)$ verwendet.

Es zeigt sich, daß wegen der geringsten Feldgröße die in der Tabelle als erster Algorithmus genannte direkte Realisierung mit den wenigsten Multiplikationen und Additionen auskommt.

Die zweite Möglichkeit benötigt im Falle eines quadratischen Feldes der Kantenlänge 64 ca. 4% mehr Additionen und Multiplikationen, wobei dieser Prozentsatz mit steigender Kantenlänge niedriger wird. Diese Tendenz gilt auch für das dritte und vierte Verfahren, so daß diese Realisierungen gegenüber dem direkten Algorithmus wegen ihrer wesentlich einfacheren Realisierbarkeit durch ein eindimensionales nichtrekursives Filter günstiger sind. Es lassen sich dann - wie bereits erwähnt - die im Abschnitt 3.2.1 eingeführten Strukturen für Teilprozessoren anwenden.

Die Tabelle zeigt, daß das dritte Verfahren gegenüber allen anderen die geringste Zahl von Multiplikationen aufweist. Dies entspricht der Ausnutzung der hohen Symmetrie des hier verwendeten Koeffizientenfeldes. Es ist somit für diesen Filtertyp bzgl. des Multiplikationsaufwandes die vorteilhafteste Realisierung.

Es bleibt nun noch das zweite mit dem vierten Verfahren zu vergleichen.

Der vierte Algorithmus, der ein Filter mit variabler Impulsantwort verwendet, weist einen gegenüber dem dritten Verfahren geringeren Anstieg der Zahl der Operationen auf, als das unter Punkt 2 in der Tabelle genannte. Dafür benötigt es jedoch einen geringfügig größeren Koeffizientenspeicher oder, und dies kann ein wesentlich gravierenderer Nachteil sein, eine höhere Zahl von Datentransporten der Koeffizienten in das Multiplizierwerk eines Teilprozessors. Damit

wird noch einmal deutlich, daß die Architektur eines Signalprozessors entscheidenden Einfluß auf die Wahl des Algorithmus hat:

- Wenn sich in einem Teilprozessor nach Bild (3.2-2) in Abschnitt 3.2.1.4 in einem Prozessorzyklus der Eingangswert x_1 (was hier einem Wert $u'(k)$ bzw. $u''(k)$ entspricht) und ein Koeffizient zu dem Multiplizierwerk transportieren lassen, so ist das Verfahren 4 günstiger, da es mit weniger Multiplikationen und damit weniger Prozessorzyklen auskommt.
- Andernfalls ist dem Algorithmus 2 der Vorzug zu geben, da durch die Konstanz der Impulsantwort die Koeffizienten nur einmal in das Multiplikationsregister geladen werden müssen und sodann unverändert belassen werden können.

3.2.6 Rekursive Bildfolgenverarbeitung

3.2.6.1 Prinzip

Aufbauend auf den in Abschnitt 3.2.5 angegebenen Möglichkeiten der 2-dimensionalen Feldverarbeitung wird nun eine Methode beschrieben, die einen in [24] angegebenen Algorithmus zur Verarbeitung von Bildfolgen realisiert. Bild (3.2-30) zeigt das Prinzip.

Dabei werden Folgen von 2-dimensionalen Datenfeldern betrachtet, wobei die Felder mit κ indiziert sind, und ihre Elemente gemäß der Vorgehensweise des vorigen Abschnitts mit den Koordinaten l_1 und l_2 angegeben werden. Ein Feldelement wird daher in den drei Dimensionen κ , l_1 und l_2 beschrieben, weshalb diese Art der Feld- bzw. Bildverarbeitung als 3-dimensionale Signalverarbeitung bezeichnet werden kann. Bild (3.2-30) zeigt die Struktur. Zunächst seien für $\kappa < 0$ sämtliche Elemente des Feldes $y(\kappa, l_1, l_2) \equiv 0$. Sodann werden für $\kappa \geq 0$ Bildfolgen $u(\kappa, l_1, l_2)$ zu dem durch

$$a(l_1, l_2) \Rightarrow \bigoplus \Rightarrow c(l_1, l_2) = a(l_1, l_2) + b(l_1, l_2)$$

$\uparrow$
 $b(l_1, l_2)$

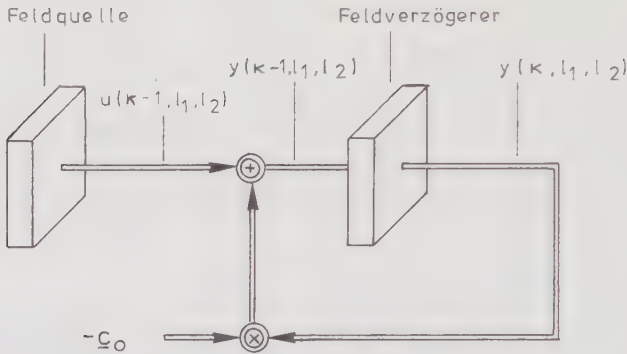


Bild (3.2-30) Struktur für einen Algorithmus zur Bildfolgenverarbeitung nach [24]

symbolisierten Addierer geleitet, der zu der Folge von Eingangsfeldern $u(k, l_1, l_2)$ die mit der Matrix $-C_0$ multiplizierten Ergebnisfelder $y(k-1, l_1, l_2)$ elementweise hinzuaddiert.

Das Datenfeld $u(k, l_1, l_2)$ und $y(k, l_1, l_2)$ wird im folgenden wieder als Vektor $\mathbf{u}(k)$ und $\mathbf{y}(k)$ dargestellt, so daß die Multiplikation $-C_0 \cdot \mathbf{y}(k)$ durch die in Abschnitt 3.2.5 erklärten Methoden zu realisieren ist.

Das in Bild (3.2-30) dargestellte System wird dann beschrieben durch

$$\mathbf{y}(k) = \mathbf{u}(k-1) - C_0 \cdot \mathbf{y}(k-1) , \quad (3.2-63)$$

3.2.6.2 Realisierung durch ein variables, eindimensionales, rückgekoppeltes nichtrekursives Filter

Als Beispiel für die Berechnung von Gleichung (3.2-63) wird das in Abschnitt 3.2.5.2c beschriebene Verfahren zur Berechnung des Produktes $-C_0 \cdot \mathbf{y}(k-1)$ angewendet. Es wird dabei vorausgesetzt, daß $-C_0$ die Form der Matrix $\underline{G}$ aus Abschnitt 3.2.5 be-

sitzt. Dann lassen sich die gleichen Modifikationsmatrizen verwenden, und aus Gl. (3.2-63) wird

$$\mathbf{y}(\kappa) = \mathbf{u}(\kappa-1) + \mathbf{M}_2''(-\mathbf{C}_0'')\mathbf{M}_1'' \mathbf{y}(\kappa-1) \quad .$$

Mit $\mathbf{M}_2'' = \mathbf{E}$

ergibt sich

$$\mathbf{y}(\kappa) = \mathbf{u}(\kappa-1) + (-\mathbf{C}_0'')\mathbf{M}_1'' \mathbf{y}(\kappa-1) \quad . \quad (3.2-64)$$

Setzt man $\mathbf{M}_1'' \mathbf{y}(\kappa) = \mathbf{y}''(\kappa)$

erhält man schließlich

$$\mathbf{y}''(\kappa) = \mathbf{M}_1'' (\mathbf{u}(\kappa-1) + (-\mathbf{C}_0'') \mathbf{y}''(\kappa-1)) \quad . \quad (3.2-65)$$

Gleichung (3.2-65) beschreibt das Gesamtsystem, das im Kern wieder durch ein eindimensionales, in k variables nichtrekursives Filter realisiert werden soll.

Deshalb werden die Vektoren $\mathbf{y}''(\kappa)$ und $\mathbf{u}(\kappa)$ wieder durch ihre Elemente dargestellt. Es ergibt sich z.B.

$$\mathbf{y}''(\kappa) \rightarrow y''(\kappa, l_1, l_2) \rightarrow y''(\kappa, k) \quad .$$

Damit kann Gleichung (3.2-64) geschrieben werden als

$$y(\kappa, k) = u(\kappa-1) - \sum_{\mu=0}^n c_0''(\mu, k) y''(\kappa-1, k+\mu) \quad . \quad (3.2-66)$$

Für den Gültigkeitsbereich von Gleichung (3.2-66) gilt das in Abschnitt 3.2.5.2c zu Gl. (3.2-60) Gesagte entsprechend.

a) Struktur

Bei der Verwirklichung von Gleichung (3.2-66) muß der Addition von $u(\kappa, k)$ zu dem Filterausgangssignal besondere Aufmerksamkeit geschenkt werden, da es um n Takte verzögert am Filterausgang erscheint. Die entsprechende Verzögerung ist deshalb auch für die Folge $u(\kappa, k)$ vorzusehen. Da für den nächsten Rekursionszyklus zur Berechnung von

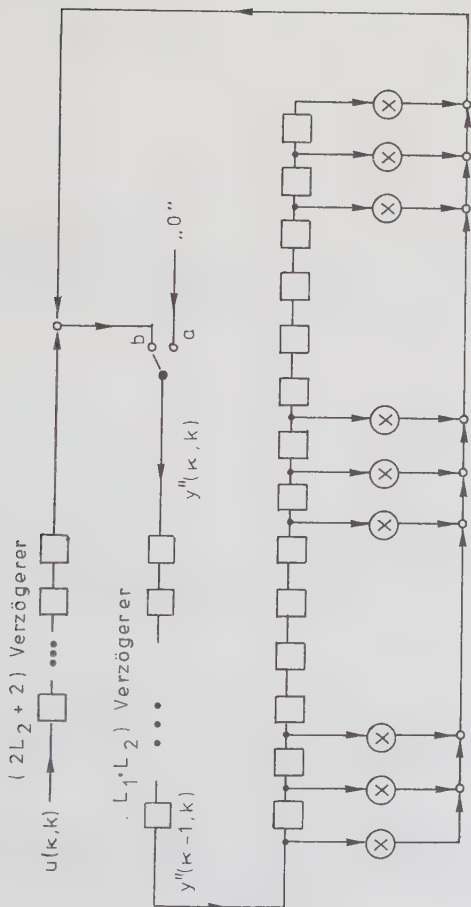
$y(k+1, k)$ die Elemente des Vektors $\mathbf{M}_1'' \mathbf{y}(k)$ benötigt werden, müssen die Ergebniswerte nach Gleichung (3.2-66) entsprechend der Forderung der Matrix $\mathbf{M}_1''$ mit Nullen erweitert werden. Dies leistet ein von k gesteuerter Schalter, der entweder den Signalweg schließt oder Nullen einfügt. Bild (3.2-31) zeigt das System, das diesen Algorithmus realisiert. Zur Berechnung eines Ergebnisvektors $\mathbf{y}(k)$ sind dabei für k insgesamt $[L_2(L_1+2)+2]$ Rechenzyklen erforderlich.

Das durch den Vektor $\mathbf{y}''(k)$ beschriebene Datenfeld befindet sich verteilt in den $L_1 L_2$ Verzögerern im Signalweg nach dem Schalter, sowie in den Speicherelementen des nichtrekursiven Filters.

b) Verwendung von Teilprozessoren

Die Implementierung des Systems nach Bild (3.2-31) auf einem programmierbaren Rechenwerk kann wieder mit Teilprozessoren gemäß dem Abschnitt 3.2.1 vorgenommen werden, bei denen lediglich Speicher hinreichend großer Kapazität zur Verfügung stehen müssen, in denen die variablen Koeffizienten $c_0(k)$ sowie die Datenpuffer für die Folge $y''(k, k)$ realisierbar sind. Dies wird in Bild (3.2-32) dargestellt.

Die Realisierung der Schalterfunktion in Bild (3.2-31) entspricht lediglich dem programmierbaren Löschen eines Speicherbereichs der Länge $(2L_2+2)$, sie stellt keine speziellen Anforderungen an die Struktur eines Teilprozessors.



Schalterstellung:

a für $k=1 \dots (2L_2+2)$

b sonst

Bild (3.2-31)

Systemstruktur zur Realisierung des rekursiven 3-dimensionalen Algorithmus nach Gleichung (3.2-64 bzw. -66). Es wird auch hier eine Impulsantwort nach Bild (3.2-27) zugrundegelegt, die Koeffizienten $c''_0(\nu, k)$ sind der besseren Übersichtlichkeit wegen nicht eingezeichnet.

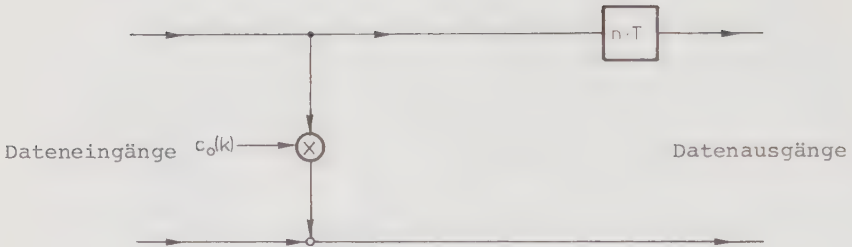


Bild (3.2-32) Geringfügig modifizierte Struktur eines Teilprozessors nach Bild (3.2-2) zur Realisierung des in Bild (3.2-31) angegebenen Algorithmus

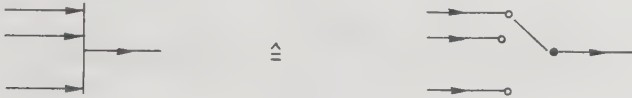
3.2.7 Zusammenfassung

In diesem Kapitel wurde versucht, für eine Auswahl von Algorithmen Teiloperationen zu finden, die durch Kaskadenschaltungen^{*)} von Teilprozessoren realisierbar sind. Dabei war es ein Hauptziel, für die unterschiedlichen Algorithmen jeweils möglichst ähnliche Strukturen der Teilprozessoren zu finden, so daß nun zusammenfassend Strukturen angegeben werden können, die die vorher gefundenen beinhalten und die spezifischen Anforderungen der hier untersuchten Algorithmen erfüllen.

*) Es ist hier unerheblich, ob die Kaskadierung durch Programmschleifen in einem Prozessor, oder durch ein Multiprozessorsystem realisiert wird.

3.2.7.1 Universeller Teilprozessor

In Bild (3.2-33) ist ein universeller Teilprozessor angegeben, der die vorher behandelten Systeme direkt realisiert, wenn z.B. durch ein Mikroprogramm die Datenpfade entsprechend den gestrichelt eingezeichneten Linien geschaltet werden. Es wird das Symbol



zur Beschreibung eines von einem Programm einstellbaren Multiplexers verwendet.

Bei dem angegebenen Teilprozessor ist die Realisierung des Speichers A problematisch, da er über zwei unabhängige Datenpfade ausgelesen und über einen beschrieben wird.

Es ist somit ein 3-Tor-Speicher notwendig. Wird er aus den üblicherweise verwendeten 1-Tor-Speichern aufgebaut, so muß er durch eine Steuerung instandegesetzt werden, 3 Speicheranforderungen in einem vorgegebenen Zeitrahmen, z.B. einem Prozessorzyklus, bedienen zu können. Dies würde einen Speicher erfordern, dessen Zykluszeit dreimal kürzer als die des Prozessors ist, wenn alle in Bild (3.2-33a...d) eingezeichneten Datenpfade in einem Prozessorzyklus realisierbar sind.

Es ist weiterhin eine Schiebelogik vorgesehen. Sie dient dazu, ohne zusätzliche Prozessorzyklen das Ergebnis des letzten Addierers um eine Bitposition zum LSB oder MSB zu schieben, oder es zu belassen. So können Normierungen vorgenommen werden, wie sie z.B. bei rekursiven Filtern (Abschnitt 3.2.2) vorkommen.

Erwähnenswert ist weiterhin, daß in Bild (3.2-33b) im Zweig x'_1 nach x_1 zwei Verzögerungsglieder vorhanden sind. Dies kommt daher, daß jeder Teilprozessor mit Ein- bzw. Ausgaberegistern versehen sein muß. Von den beiden Verzögerungs-

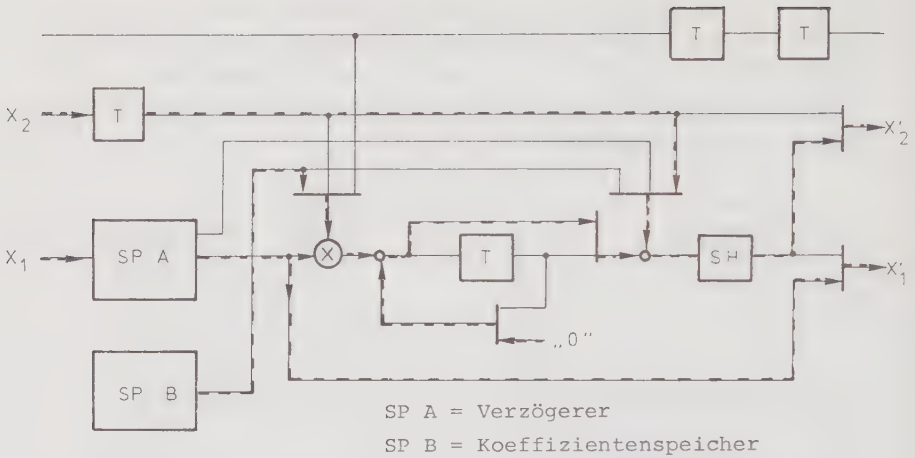


Bild (3.2-33a) Datenwege für die Realisierung eines nicht-rekursiven Filters nach Bild (3.2-2)

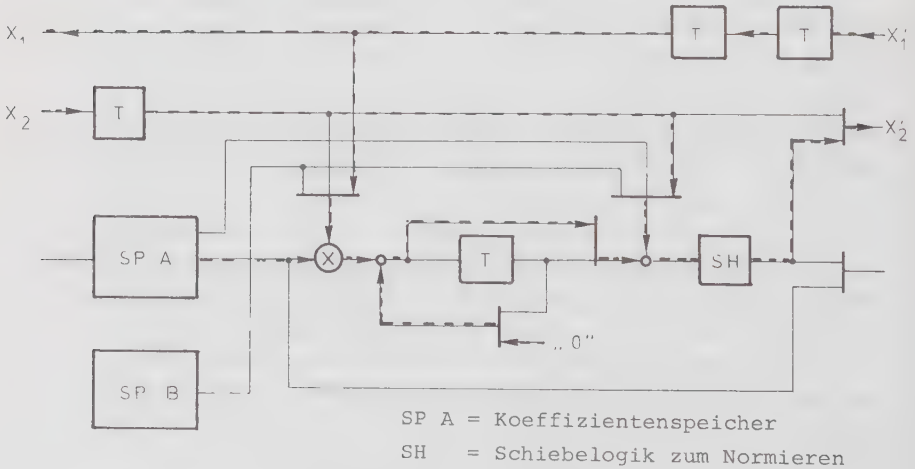


Bild (3.2-33b) Rekursiver Teil eines rekursiven Filters nach Bild (3.2-9)

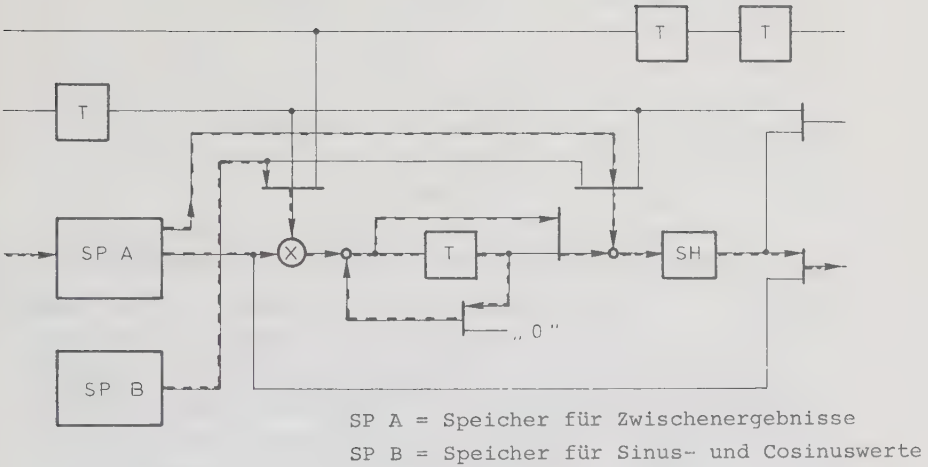


Bild (3.2-33c) FFT-Butterfly nach Bild (3.2-15),
2.Schritt als Beispiel

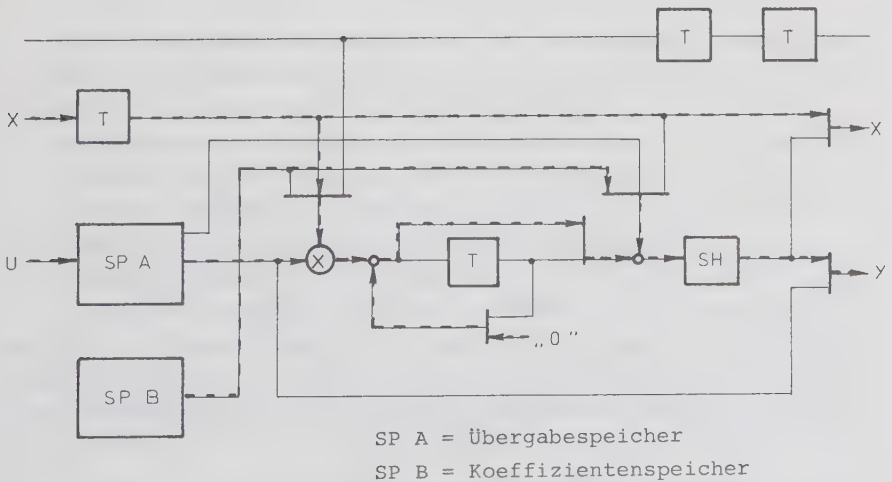


Bild (3.2-33d) Potenzreihenentwicklung nach Bild (3.2-19)

gliedern dient eines dieser Aufgabe, das andere realisiert die vom Filteralgorithmus geforderte Signalverzögerung.

Werden mehrere Teilprozessoren nach Bild (3.2-33) zu einem Multiprozessorsystem zusammengeschaltet, so muß am Anfang und am Ende der Kette jeweils ein Ein/Ausgabeprozessor den Datenverkehr mit der Peripherie wie z.B. Analog-Digital- und Digital-Analog-Wandlern übernehmen.

3.2.7.2 Teilprozessor mit erhöhter Zahl von Rechenwerken

Das in 3.2.7.1 angegebene Konzept eines Teilprozessors eignet sich insbesondere für die Realisierung eines Multiprozessor-systems, bei dem benachbarte, linear angeordnete Prozessoren in einem streng synchronen Taktrahmen Daten austauschen. Es wird nun eine Prozessorstruktur angegeben, die eine höhere Zahl von Rechenwerken bereitstellt und sich speziell für Filteralgorithmen bei Mehrfachausnutzung eignet. Dabei wird ausgegangen von der Beobachtung, daß bei den hier behandelten Algorithmen die Produktsummenbildung

$$y = \sum_{\rho} a_{\rho} x_{\rho} \quad (3.2-67)$$

eine zentrale Operation ist. Dies kommt deutlich z.B. in Bild (3.2-3) zum Ausdruck, wo die kaskadierten Teilprozessoren zur Berechnung des Ausgangswertes $y(k)$ eines nichtrekursiven Filters direkt Gleichung (3.2-67) realisieren. Entsprechendes gilt auch für den rekursiven Filteralgorithmus nach Bild (3.2-5) bzw. (3.2-6), wo die Summe von 5 Produkten gebildet wird, um den Ausgangswert eines Filterblockes zu berechnen. Bild (3.2-13) zeigt, daß auch zur Realisierung des Butterflyalgorithmus bei der schnellen Fouriertransformation die Bildung von Produktsummen die wesentliche Rechenoperation ist. Dies ist auch aus Gleichung (3.2-34) ersichtlich, wo zur Berechnung eines Elementes des Vektors $\mathbf{x}_{\mu}$ eine Zeile der Matrix $\mathbf{A}$ mit dem Vektor $\mathbf{x}_{\mu-1}$ multipliziert werden muß.

Deshalb wird nun ein Teilprozessor eingeführt, der die Gleichungen

$$y_1 = \sum_{\rho=1}^3 a_{\rho} x_{\rho}$$

$$y_2 = \sum_{\mu=1}^3 a_{\mu} x_{\mu}$$

$$y_3 = y_1 + y_2 \quad (3.2-68)$$

direkt berechnet. Seine Struktur ist in Bild (3.2-34) angegeben. Sie realisiert zwei Produktsummen über jeweils drei und eine über sechs Glieder.

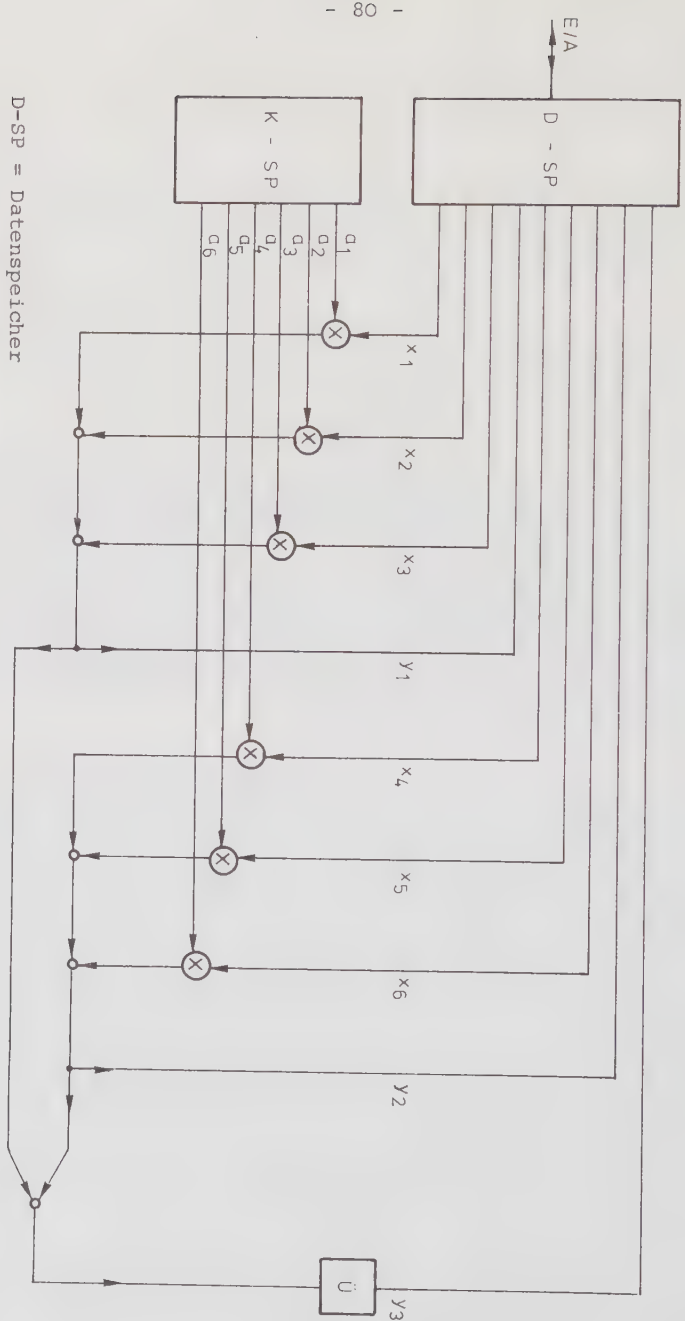
Eine interessante Eigenschaft dieses Rechenwerks ist darin zu sehen, daß es bis zu 6 Teilprozessoren nach Bild (3.2-34) in einem kompakten System realisiert, denn es führt 6 Multiplikationen und 5 Additionen in einem Rechenzyklus aus. Es stellt somit einen Mittelweg zwischen dem Einsatz einer Vielzahl von autonomen Teilprozessoren und der Verwendung nur eines einzelnen Multiplikations- und Addierwerkes im programmgesteuerten Multiplexbetrieb dar.

Das Problem des Mehrfachzugriffs auf den Speicher stellt sich bei der in Bild (3.2-34) angegebenen Struktur in noch wesentlich schärferer Form als bei der zuvor angegebenen Struktur eines Teilprozessors. Es sind hier 6 Koeffizienten a_v , 6 Variable x_v und bis zu 3 Ergebniswerte y_v in einen Rechenzyklus der Arithmetischen Einheit zu transportieren.*)

Dies würde zunächst einen Speicher erfordern, dessen Zykluszeit 15 mal kleiner als die des Rechenwerkes ist.

Geht man von der realistischen Annahme einer Rechenzeit von 2 μ sec aus, so würden für einen Datentransport 133 nsec bleiben. Um diese außerordentlich hohe Anforderung an den Speicher zu mildern, sind in Bild (3.2-34) zwei Speicherblöcke vorgesehen, von denen der eine die Koeffizienten, der andere die Daten enthält. Somit liegt die geforderte Zykluszeit für den Daten-

*) Dabei wird vorausgesetzt, daß die Datentransporte den Betrieb des Rechenwerkes nicht stören. Dies kann durch das im 5. Kapitel beschriebene Pipelineverfahren gewährleistet werden.



D-SP = Datenspeicher
 K-SP = Koeffizientenspeicher
 ü = Überlauflogik

Bild (3.2-34)

Struktur einer arithmetischen Einheit, die die Rechenleistung von bis zu 6 Teilprozessoren nach Bild (3.2-33) bietet. Die Datenwege werden hier durch zum Datenspeicher realisiert.

speicher bei über 200 nsec, was entweder zur Vereinfachung des Systems, oder zur Erhöhung der Reserve ausgenutzt werden kann.

Die im Abschnitt 3.2.2.2 angegebene Überlaufkennlinie nach Bild (3.2-8) ist in der arithmetischen Einheit für die Summe y_3 vorgesehen, da sie das Ausgangssignal eines rekursiven Filterblocks 2. Grades gemäß (3.2-5) und (-6) ist.

In den nun folgenden Kapiteln werden, speziell unter dem Aspekt des Einsatzes von Mikroprozessoren und hochintegrierten Multiplizierbausteinen, Möglichkeiten zur Realisierung der gefundenen Teilprozessorstrukturen angegeben.

Um die Flexibilität der zu beschreibenden Signalprozessoren nicht unzulässig einzuschränken, werden im nächsten Kapitel einige allgemeine Anforderungen zusammengestellt, deren Realisierung in einem Signalprozessor gewährleistet, daß er sich als programmierbares System auch für andere als die bisher angegebenen Algorithmen eignet.

4. Spezielle Anforderungen an Signalprozessoren

Ziel des letzten Kapitels war das Auffinden von Programm- bzw. Hardwarestrukturen, die den ausgewählten Algorithmen angepaßt sind. Es sollen nun darüberhinaus einige Forderungen formuliert werden, die konkret in den Entwurf eines Signalprozessors eingehen. Sie sind jedoch in der Mehrzahl nur dann realisierbar, wenn keine Mikroprozessoren mit festgelegter Architektur und festgelegtem Befehlssatz zur Anwendung kommen.

4.1 Adressierung

Ein wesentliches Leistungsmerkmal jedes Rechners sind seine Adressierungsmöglichkeiten. Die Bereitstellung von einer oder mehreren Adressen ist bei jeder Operandenverknüpfung bzw. bei jedem Datentransport notwendig. Es wird hier nur die Adressierung auf Maschinenbefehlsebene betrachtet.

4.1.1 Operandenadressierung

Geht man von der in Gleichung (3.2-67) beschriebenen Produktsammenbildung als immer wiederkehrende Basisoperation aus, so sind zu deren Realisierung zunächst zwei Operationen von besonderem Interesse:

- Es müssen zur Ausführung des Multiplikationsbefehles

MUL A, X, Y

die Datenworte A und X aus dem Speicherwerk geholt und miteinander multipliziert werden, danach muß das Produkt in die Speicherstelle Y transportiert werden.

- Im nächsten Schritt wird mittels des Befehls

ADD Y, S, S

das Produkt Y zum Inhalt der Speicherstelle S hinzuaddiert und danach die Summe in S abgelegt, wobei deren früherer Inhalt verloren gehen darf. Dies realisiert die Akkumulationsfunktion.

Diese Detailbeschreibung der Operandentransporte zeigt, daß der Multiplikationsbefehl ein 3-Adreßbefehl ist, denn A und X sind bei der Realisierung von Gleichung (3.2-67) Koeffizienten bzw. Zustandsvariable, die bei nachfolgenden Multiplikationszyklen wiederverwendet werden.

Die Ausführung der Akkumulation ist dagegen mit der Angabe zweier Adressen beschreibbar.

Die Zahl der Adressen in einem Maschinenbefehl beeinflusst direkt den Hardwareentwurf eines Prozessors. Zur Sicherstellung einer gewissen Architekturhomogenität wird man deshalb für alle Befehle die gleichen Adressierungsmöglichkeiten vorsehen. Demnach ist es angemessen, einen Signalprozessor als 3-Adreßmaschine zu entwerfen.

4.1.2 Indizierte Adressierung

Bei einer indizierten Adressierung wird die physikalische Speicheradresse als Summe aus dem Inhalt eines Indexregisters und einem im Maschinenbefehl angegebenen Displacement gebildet. Dieser Adressierungsmodus eignet sich speziell für die Implementierung der Moduloarithmetik, wie sie in Kapitel 3.2.1.5 beschrieben worden ist. Eine erhebliche Steigerung der Leistungsfähigkeit bei der Adreßgenerierung läßt sich dadurch erreichen, daß nach der Abarbeitung eines Transportbefehles das Indexregister automatisch inkrementiert bzw. dekrementiert wird. Ein Datentransportbefehl hat dann z.B. die Form

TRA BASIS (Indexregister, +1, Mod n), Ziel ,

wenn Daten von einer Quelladresse zu einer Zieladresse transportiert werden sollen. Nach Ausführung des Transports wird das Indexregister um Eins erhöht, wobei diese Addition in Moduloarithmetik bezüglich dem Inhalt n eines zugeordneten Moduloregisters gerechnet wird. Zur Verdeutlichung der Zusammenhänge wird ein Beispiel angegeben, das die Verzögererkette in einem Teilprozessor nach Bild (3.2-32) für $n = 4$ realisiert:



Das Programm besteht im wesentlichen aus fünf Befehlen, von denen lediglich drei die eigentliche Verzögerungsoperation bewerkstelligen:

1. Lade Moduloregister mit 4
2. Lösche Indexregister
3. BEG: TRA BASIS (Indexregister), $u(k-4)$
 - Transport des Datenwortes von der Adresse Basis plus Inhalt vom Indexregister an die Ausgabe
4. TRA $u(k)$, BASIS (Indexregister, +1, Mod 4)
 - Transport eines Eingangsdatenwortes in den Speicher, anschließendes Inkrementieren des Indexregisters in Moduloarithmetik
5. Springe unbedingt nach BEG

Hierbei kann die Verzögererkette über die Basisadresse BASIS an einer beliebigen Stelle des Datenspeichers aufgebaut werden.

Es sei hier lediglich angedeutet, daß zur Beschleunigung von Programmschleifen eine automatische Schleifenverwaltung mit einem Indexregister realisierbar ist, wenn ihm ein Offsetregister zugeordnet wird.

Nach bzw. vor jeder Inkrementierung des Indexregisters läßt sich durch Vergleich der Registerinhalte bei Übereinstimmung eine Skipbedingung ableiten, so daß die automatische Adreßgenerierung gleichzeitig die Schleifenverwaltung übernimmt.

Es läßt sich dann z.B. ein Datenfeld der Länge n durch folgendes kurze Programm im Speicher verschieben:

1. Lösche Indexregister
2. Lade Offsetregister mit n
3. A: TRA BASIS 1 (Indexregister), BASIS 2 (Indexregister, +1, skip falls {Indexregister} = {Offsetregister})
4. Springe unbedingt nach A

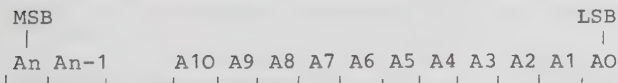
Dabei wird der Blocktransfer von Befehl 3 und 4 ausgeführt.

4.1.3 Bitreversal

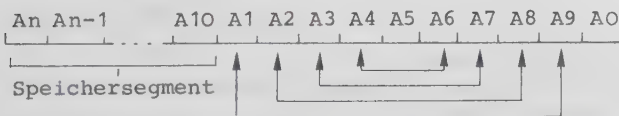
Bei der Berechnung der schnellen Fouriertransformation z.B. gemäß dem Signalfluß nach Bild (3.2-11) sind die Elemente des Ausgangsvektors **u** nicht mit steigendem Index geordnet, sondern in Adressen, in deren Binärdarstellung die Bit vertauscht sind (z.B. [21]).

Dies bedeutet, daß beim inkrementellen Auslesen des Speicherbereiches in dem **u** abgelegt ist, die Adressen umgerechnet werden müssen. In einem Signalprozessor ist daher ein Adressierungsmodus, der das "Bitreversal" leistet, von besonderem Interesse. Für die häufig benutzte Transformationslänge von 512 komplexen Werten bietet sich bei einer Adresse, deren Binärdarstellung (n+1) Bit benötigt, folgendes Muster an:

Originaladresse:



Bitreversed Adresse:



Diese Form der (Hardware-)Realisierung bietet zwei Vorteile:

- Durch Festhalten des Bit A0 ist das Abspeichern von Real- und Imaginärteil in geraden bzw. ungeraden Adressen (oder umgekehrt) leicht möglich.

- Da die Adreßbits A10 bis An unverrückt bleiben, lassen sich mit ihnen Speichersegmente anwählen, in denen die Datenfelder für den FFT-Algorithmus abgespeichert sind.

Es sei noch erwähnt, daß sich die angegebene Methode des Bitreversal auch für kürzere Transformationslängen eignet, wenn im Speicher Lücken zugelassen werden. Wird in einem Signalprozessor die FFT nur in einer Struktur, z.B. "Bitreversal am Ausgang" wie in Bild (3.2-11) gerechnet, so genügt es, diesen Adreßmodus nur für die Quelladresse in Datentransportbefehlen vorzusehen.

4.2 Synchronisationsmechanismen

Speziell bei Anwendungen in der Echtzeitsignalverarbeitung soll ein Signalprozessor durch einfache Methoden mit einem extern vorgegebenen Taktraster synchronisierbar sein. Dazu werden zwei Möglichkeiten beschrieben.

4.2.1 Taktsynchronisation

Wird mit einem Signalprozessor z.B. eine Filterung durchgeführt, so läßt sich das realisierende Programm im wesentlichen durch 3 Programnteile charakterisieren:



Vorbereitung: z.B. Laden der Koeffizienten, Setzen der Index- und Moduloregister usw.

Synchronisation: Warten auf einen Abtasttakt

Filtern: Ausführung des Algorithmus

+ Rücksprung: Aufrechterhaltung des Programmlaufes.

Während der Synchronisationsphase wartet der Prozessor auf einen Impuls, der anzeigt, daß z.B. ein angeschlossener Analog-Digital-Wandler einen neuen Abtastwert geliefert hat. Wird hierzu ein Synchronisationsflipflop verwendet, das vom Prozessor abgefragt und wenn es gesetzt war, gelöscht wird, so ist hierzu eine eigene Programmschleife mit 3 bis 4 Maschinenbefehlen notwendig, so daß zur Abfrage ein Zeitaufwand in der Größenordnung von 1 ... 10 μ sec erforderlich ist (z.B. [25]). Da sich die Synchronisationsphase in der Programmschleife befindet, bedeutet dies einen Verlust für die maximal mögliche Abtastfrequenz des Systems.

Einen wesentlich günstigeren Synchronisationsmechanismus erhält man, wenn durch einen Maschinenbefehl der Takt des Rechners abgeschaltet und durch einen externen Impuls wieder eingeschaltet wird. Die prinzipielle Wirkungsweise zeigt Bild (4-1).

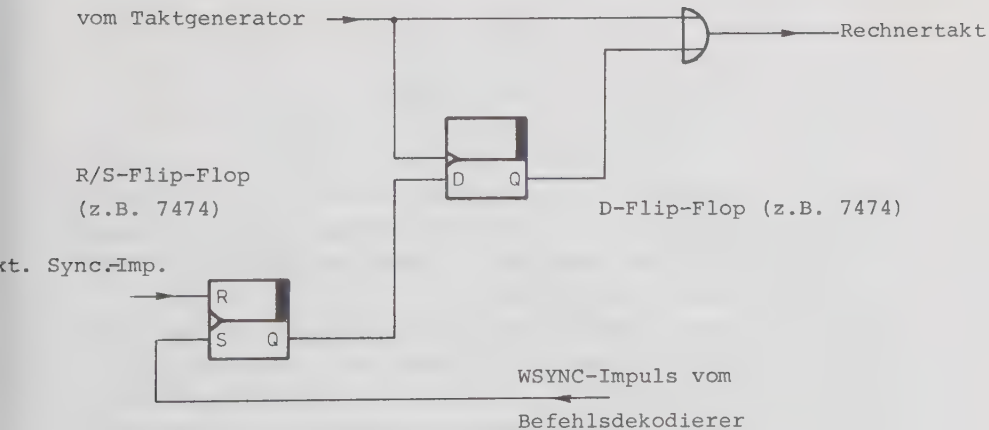


Bild (4-1) Prinzipielle Schaltung zur Realisierung des Synchronisationsbefehles

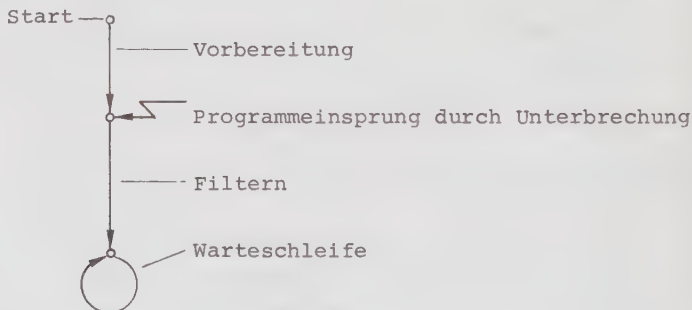
Mit einem WSYNC-Impuls (= "Warte auf Synchronimpuls"), den der Befehlsdekodierer bei Auftreten des entsprechenden Maschinen-

befehles liefert, wird das R/S-Flip-Flop gesetzt und somit der Takt über das D-Flip-Flop und das ODER-Gatter gesperrt. Dadurch wird die Programmbearbeitung unterbrochen, bis durch einen externen Synchronimpuls die Wartefunktion wieder rückgängig gemacht wird.

Diese Methode der Taktsynchronisierung benötigt lediglich einen Maschinenbefehl.

4.2.2 Unterbrechungssynchronisation

Eine zweite Methode der Synchronisation erhält man durch Anwendung der Unterbrechungstechnik. Dabei ergibt sich eine gegenüber dem vorigen Abschnitt modifizierte Darstellung des Programmablaufes:



Bei Auftreten eines externen Synchronimpulses wird das Rechenwerk gezwungen, bei der Programmbearbeitung die Warteschleife zu verlassen und das Programm bei einer vereinbarten Adresse (Einsprungsadresse) fortzuführen.

Bei dieser Unterbrechungstechnik ist, entgegen den bei Universalrechnern üblichen Verfahren, dafür zu sorgen, daß der Inhalt des Programmzählers *n i c h t* als Rücksprungsadresse in einen Stackbereich des Speichers geladen wird, da dies wegen der ständig erscheinenden Synchronimpulse den Speicher auffüllen und damit blockieren würde.

4.3 Datenverknüpfungen

Beim Entwurf des Befehlssatzes eines Signalprozessors ist darauf zu achten, daß neben den arithmetischen Datenverknüpfungsbefehlen wie Addition, Multiplikation, Division und Produktsummenbildung auch die bitweise logische Verknüpfung zweier Datenworte entsprechend den Möglichkeiten eines Universalrechners implementiert werden. Nur dadurch sind z.B. arithmetische Wortlängenverkürzungen oder Statusbitabfragen möglich.

Der Befehlssatz eines universellen Signalprozessors unterscheidet sich hierbei nicht prinzipiell von dem eines Universalrechners und wird deshalb hier nicht detailliert beschrieben.

5. Möglichkeiten für die Architektur von Signalprozessoren bei Verwendung neuerer Bauelemente

Im Hinblick auf eine möglichst hohe Datendurchsatzrate werden hier nur solche Signalprozessorkonzepte angegeben, die mit bipolaren hochintegrierten Bausteinen aufgebaut werden können. Es werden also keine Systeme betrachtet, in denen MOS-Mikroprozessoren eine zentrale Rolle bei der Algorithmenrealisierung spielen. Es sind zwar "Array-Computer" bekannt geworden (z.B. [26]), in denen versucht wird, die geringe Rechengeschwindigkeit eines MOS-Prozessors (z.B. des 8080) durch eine sehr hohe Anzahl von Rechenwerken wett zumachen, doch scheint diese Vorgehensweise - zumindest im Bereich der digitalen Echtzeit-Signalverarbeitung - erst dann sinnvoll zu werden, wenn noch wesentlich leistungsfähigere Mikroprozessorkomponenten verfügbar sind. Dies soll mit folgender Überlegung belegt werden:

Der MOS-Mikroprozessor 8080 kann 8-Bit breite Datenworte verarbeiten. Zur Ausführung einer 16x16-Bit-Multiplikation ist deshalb eine zeitaufwendige Doppelwortarithmetik notwendig, so daß die Multiplikationszeit bei 400 μ sec liegt. Der Einsatz von z.B. 100 Rechnern würde eine mittlere Multiplikationszeit von 4 μ sec ergeben und damit aber immer noch um den Faktor 40 über der von einem schnellen Parallelmultiplizierer benötigten Rechenzeit liegen. Probleme des Datentransportes sind hierbei nicht berücksichtigt. Die mechanischen Abmessungen des Multipliziererschips entsprechen dabei etwa denen eines einzelnen MOS-Mikroprozessors.

Diese Verhältnisse können durch den Einsatz von MOS-Mikroprozessoren, die 16 Bit verarbeiten und Maschinenbefehle für die Multiplikation zur Verfügung stellen, zwar verbessert werden, doch liegt auch bei diesen Rechnern die Multiplikationszeit für 2'er-Komplementzahlen z.T. noch erheblich über 10 μ sec, so daß auch hier eine sehr hohe Zahl von Rechnern nötig ist, um die Leistungsfähigkeit eines bipolaren Parallelmultiplizierers zu erreichen.

Dabei sind die Steuerungsprobleme noch gar nicht angesprochen, die durch die Zusammenarbeit vieler Prozessoren entstehen, insbesondere dann, wenn sie nicht durch einfache Kaskadierung - wie im 3. Kapitel beschrieben - gekoppelt werden können.

In diesem Kapitel werden zwei Signalprozessorsysteme beschrieben, die in bipolarer Technik realisiert sind und sich speziell für die im 3. Kapitel behandelten Aufgaben der Signalverarbeitung eignen. Das zunächst beschriebene Minimalsystem sowie der universelle Signalprozessor wurden bereits aufgebaut und erprobt.

5.1 Pipelineprinzip

Pipelineverfahren werden bei der Realisierung digitaler Systeme häufig angewendet (z.B. [19, 27, 28]). Da die in diesem Kapitel behandelten Prozessorstrukturen ausnahmslos mit Pipelineverfahren arbeiten, sei hier zunächst das Prinzip des Pipelining allgemein dargestellt.

5.1.1 Problemstellung

Es wird ein System zur Verarbeitung binärer Daten betrachtet, wie es in Bild (5.1-1) dargestellt ist. Die dabei verwendeten Register seien verzögerungsfrei.

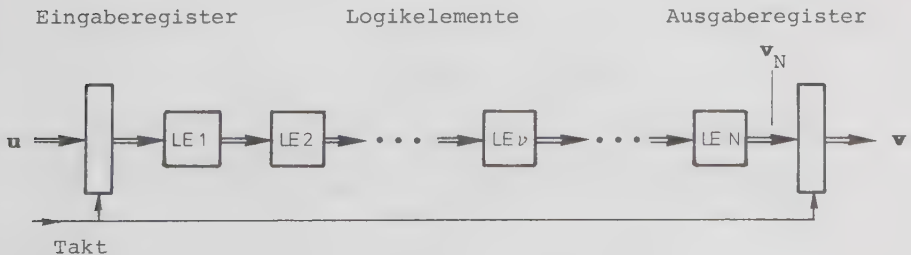


Bild (5.1-1) Lineare Anordnung von Logikelementen zur Datenverarbeitung

Ein Logikelement LE_v erzeugt aus einem Eingangsvektor $\mathbf{u}_v$ der Dimension $(e+1)$ einen Ausgangsvektor $\mathbf{v}_v$ der Dimension $(a+1)$, Bild (5.1-2).

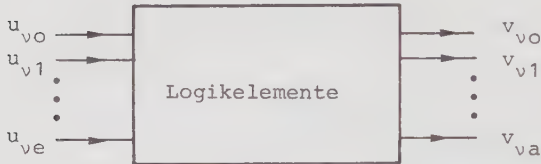


Bild (5.1-2) Logikelement zur Umrechnung des Eingangsvektor $\mathbf{u}_v$ in den Ausgangsvektor $\mathbf{v}_v$

Der μ -te Ausgang des v -ten Logikelementes reagiere mit einer Verzögerungszeit von $\leq \tau_{v\mu}$ auf eine beliebige Änderung des Eingangsvektors $\mathbf{u}_v$, wobei $\tau_{v\mu}$ den Maximalwert in Abhängigkeit von allen möglichen Änderungen von $\mathbf{u}_v$ bezeichnet.

Maßgebend für die erreichbare Geschwindigkeit ist dann die maximale Zeit der v -ten Stufe, die für den "langsamsten" Ausgang bei ungünstigstem Eingangsvektor nötig ist. Es gilt also

$$\tau_{\text{ges}} = \sum_{v=1}^N \max_{\mu} \{\tau_{v\mu}\} \quad (5.1-1)$$

Das Ausgangsregister (Bild (5.1-1)) darf sinnvollerweise erst dann getaktet werden, wenn sämtliche Elemente des Signalvektors $\mathbf{v}_N$ stabil geworden sind. Deshalb gilt

$$f_{\text{Takt max}} = \frac{1}{\tau_{\text{ges}}} = \frac{1}{\sum_{v=1}^N \max_{\mu} \{\tau_{v\mu}\}} \quad (5.1-2)$$

5.1.2 Pipelinelösung

Bei der Pipelinelösung werden zwischen die Logikelemente weitere Register eingefügt und zwar so, daß zwischen jeweils zwei Registern eine möglichst gleich große maximale Verzögerungszeit auftritt, Bild (5.1-3).

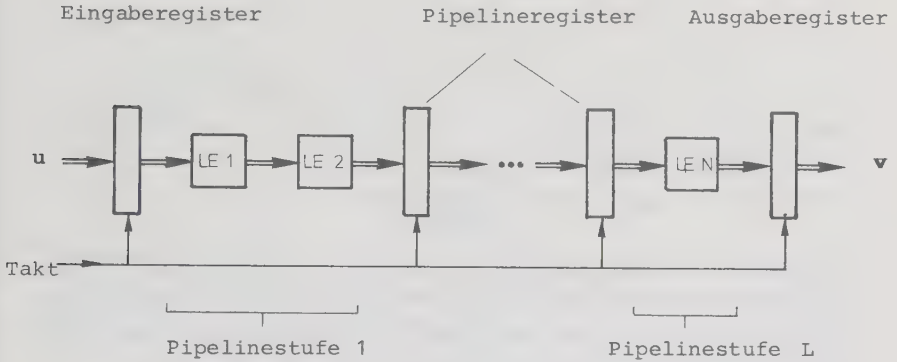


Bild (5.1-3) Pipelineanordnung

Dabei gilt für die l -te Pipelinestufe

$$\tau_l = \text{Max}_\mu \{ \tau_{l\mu} \} \quad . \quad (5.1-3)$$

Da die langsamste Pipelinestufe den Takt bestimmt, ergibt sich

$$f_{\text{Takt max}} = \frac{1}{\text{Max}_{l, \mu} \{ \tau_{l\mu} \}} = \frac{1}{\text{Max}_l \{ \tau_l \}} \quad . \quad (5.1-4)$$

Um bei einer vorgegebenen Pipelinetaktfrequenz mit einer möglichst geringen Zahl von Pipelinestufen auszukommen, soll

$$\text{Max}_l \{ \tau_l \} \approx \text{Max}_\mu \{ \tau_{l\mu} \} \quad \forall l \quad (5.1-5)$$

angestrebt werden, d.h. die Verzögerungen in jeder Stufe sollten annähernd gleich sein.

Ist diese Bedingung exakt erfüllt, so liegt die maximale Pipelinetaktfrequenz nach Gleichung (5.1-4) um das L -fache höher als die Taktfrequenz nach Gleichung (5.1-2).

5.1.3 Theoretische Geschwindigkeitsgrenze

Es läßt sich zeigen (z.B. [29]), daß sämtliche Logiktabellen mit der NAND- bzw. NOR-Funktion realisierbar sind. Deshalb kann der Term

$$\text{Max}_{l, \mu} \{ \tau_{l\mu} \}$$

in Gleichung (5.1-4) dadurch minimiert werden, daß in jeder der L Pipelinestufen nur eine einstufige NAND- bzw. NOR-Gatterlogik angewendet wird. Dann ergibt sich aus Gleichung (5.1-4)

$$f_{\text{Takt max}} = \frac{1}{\tau_{\text{Gatter}}} \quad .$$

Berücksichtigt man nun noch die bei realen Registern notwendige Datenvorbereitungszeit τ_s und ihre Verzögerungszeit τ_d , ergibt sich für die maximal mögliche Taktfrequenz jedes digitalen Systems

$$f_{\text{Max}} = \frac{1}{\tau_{\text{Gatter}} + \tau_s + \tau_d} \quad . \quad (5.1-6)$$

Diese Grenze ist unabhängig von der inneren Struktur des Systems und kann durch keine Maßnahmen der Architektur überschritten werden. Sie ist gleichzeitig die maximal mögliche Abtastfrequenz für digitale Systeme.

Die in (5.1-6) angegebenen Verzögerungszeiten sind technologieabhängig, z.B. ergibt sich für die Schottky-TTL-Technik

$$f_{\text{Max}} = \frac{1}{(5 + 3 + 9)\text{nsec}} \approx 59 \text{ MHz} \quad .$$

5.2 Zellrechnerstruktur

Ausgehend von der im Abschnitt 3.1 beschriebenen Möglichkeit der Kaskadierung von Teilprozessoren und des im letzten Abschnitt beschriebenen Prinzips des Pipelineverfahrens soll nun ein zellulares Rechnersystem beschrieben werden, von dem ein vereinfachtes Minimalsystem bereits aufgebaut und getestet wurde [30, 31].

5.2.1 Teilprozessorstruktur

Die Struktur des Teilprozessors realisiert im Kern den in Abschnitt 3.2.7.1 angegebenen universellen Teilprozessor, der hier um zusätzliche Datenpfade und Register erweitert wird. Bild (5.2-1) zeigt seinen Aufbau.

Die Architektur des Systems berücksichtigt zwei Forderungen:

- Sie soll an die innere Struktur eines handelsüblichen hochintegrierten akkumulierenden Multiplizierers, sowie an einen mikroprogrammierbaren Mikroprozessor angepaßt sein.
- Auch ein einzelner Teilprozessor soll unabhängig von der Möglichkeit zur Kaskadierung mehrerer Systeme imstande sein, die Aufgaben einer Reihe von Prozessoren zu erfüllen.

Es werden im folgenden lediglich die wesentlichen Aspekte der möglichen Datenpfade beschrieben, wobei es sich aus Platzgründen verbietet, eine Analyse sämtlicher nichtredundanter, gleichzeitig schaltbarer Wege vorzunehmen.

Weiterhin wird hier auf die Beschreibung einer geeigneten Mikroprogrammsteuerung verzichtet, es sei lediglich auf [29, 30, 32, 33, 34] verweisen. Fragen der Adreßgenerierung werden ausführlich im Abschnitt 5.3 behandelt.

5.2.1.1 Minimalsystem

Die in Bild (5.2-1) mit durchgezogenen Linien gezeichneten Datenwege beschreiben ein Minimalsystem des Teilprozessors, das imstande ist, z.B. die in Bild (3.2-2 und -9) angegebenen Teilprozessorstrukturen - auch in programmkaskadierter Form -

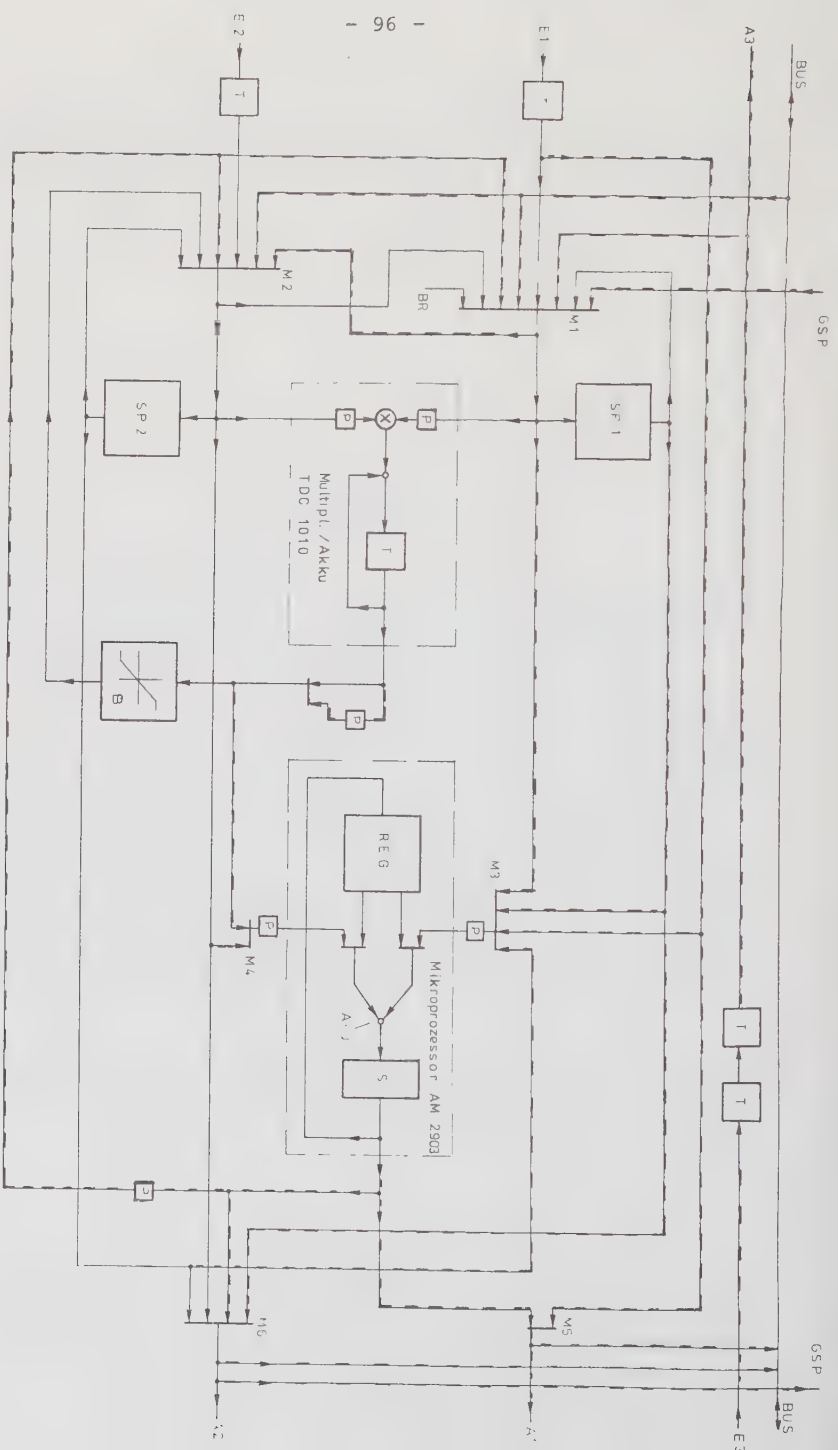


Bild 15.2.1

Struktur eines universellen Teilprozessors (P - Pipelineregister)

selbständig zu realisieren.

Es besteht im wesentlichen aus den beiden Eingabekanälen E1 und E2, sowie dem Ausgabekanal A2, den beiden Speichern SP1/SP2, und dem akkumulierenden Multiplizierer mit zugehöriger Begrenzerlogik zur Realisierung der Überlaufkennlinie nach Abschnitt 3.2.2.2. Die Datenpfade werden durch die Multiplexer M1 und M2 geschaltet.

Der Speicher SP1 eignet sich insbesondere als Koeffizientenspeicher bei der Produktsummenberechnung. Er kann vom Eingabekanal E1 oder vom Befehlsregister BR ^{*)} über den Multiplexer M1 vorab mit Koeffizienten geladen werden, die dann während der Algorithmenberechnung zum Multiplizierer geleitet werden.

Für das Abspeichern von Zwischenergebnissen (bzw. Zustandsvariablen) ist der Speicher SP2 vorgesehen. Die Datenrückführung aus dem Akkumulator über die Begrenzerlogik B führt durch den Multiplexer M2 direkt zu dem Speicher SP2, sowie zu dem Multiplizierer. Da der Ausgang von M2 mit einem Eingang von M6 verbunden ist, läßt sich dabei im gleichen Zyklus auch eine Datenausgabe realisieren.

Die Anordnung des Eingangs von SP2 parallel zum Multiplizierer, bringt den Vorteil mit sich, daß ein neuer Eingangswert vom Eingabekanal E2 gleichzeitig in den Multiplizierer und den Zustandsvariablenpeicher in SP2 geleitet werden kann.

Durch die Verbindung der beiden Eingabekanäle mit den beiden Speichern und den Multiplizierereingängen lassen sich zwei Signalabtastfolgen simultan in den Prozessor leiten, so daß z.B. die Modulation direkt möglich ist.

Allerdings sind dabei zwei Prozessorzyklen erforderlich, da der Akkumulatorausgang in diesem Minimalsystem nur über den Multiplexer M2 mit dem Ausgangskanal A2 verbunden ist.

Es sei noch angemerkt, daß sich eine hardwaremäßig realisierte Moduloarithmetik für die Adressierung der Zustandsvariablen

*) Das Befehlsregister befindet sich in der hier nicht behandelten Programmsteuerung

im Speicher SP2 dann vermeiden läßt, wenn die Speicherzykluszeit halb so lang wie die Rechenzeit des Multiplizierers ist. Dann lassen sich die Datenworte nach ihrem Auslesen durch Transport in die benachbarte Speicherstelle "verschieben". Dies gilt natürlich nur, wenn pro Multiplikationszyklus nur ein Datenwert weitergeschoben werden muß. In [30] wird dies detailliert beschrieben.

Der in Bild (5.2-1) angegebene monolitische akkumulierende Multiplizierer [35] läßt sich durch Mikrooperationsbits so steuern, daß die Akkumulationsfunktion unwirksam wird.

5.2.1.2 Erweitertes System

Die in Bild (5.2-1) gestrichelt eingezeichneten Datenpfade realisieren zusammen mit denen des Minimalsystems im wesentlichen die in Abschnitt 3.2.7.1 angegebene Struktur eines universellen Teilprozessors. Dadurch ist gewährleistet, daß sich dieses System zur Realisierung der im 3. Kapitel beschriebenen Algorithmen eignet.

Im folgenden werden die gegenüber Bild (3.2-33) hinzugekommenen Datenwege kurz erläutert. Die entscheidende Erweiterung besteht dabei in der Anwendung eines bipolaren, mikroprogrammierbaren Mikroprozessors AM 2903 [36]. Von ihm sind in Bild (5.2-1) lediglich die hier interessierenden Teile eingezeichnet. Er unterscheidet sich von dem weitverbreiteten Vorgängertyp AM 2901 im wesentlichen durch den symmetrischen Aufbau der beiden Multiplexer, die das gleichzeitige Einlesen zweier externer Datenworte in die Arithmetik- und Logikeinheit (ALU) gestatten. Deshalb genügt ein Maschinenzklus zur Verarbeitung externer Daten.

Weiterhin stellt dieser Mikroprozessor einen internen Registersatz mit einer Kapazität von 16 Worten zur Verfügung, so daß z.B. Wortlängenverkürzungen mit einer in einem Register enthaltenen Maske durchführbar sind. Die im Mikroprozessor enthaltene Steuerung erlaubt darüberhinaus auch die Realisierung von Divisionsalgorithmen, wozu allerdings mehrere Maschinenzyklen notwendig sind.

An den Ausgang der ALU des Mikroprozessors schließt sich ein Schiebemultiplexer S an, der entsprechend der Anordnung in Bild (3.2-33) für Normierungszwecke benutzt werden kann.

Der Mikroprozessor wird über die Multiplexer M3 und M4 mit Daten versorgt. Dabei sind die Datenwege so ausgewählt, daß sich sämtliche Kombinationen zweier Datenquellen des Systems als Operanden in den Mikroprozessor leiten lassen.

In Bild (5.2-1) ist am oberen Bildrand eine mit BUS bezeichnete Datenleitung eingezeichnet. Hierauf, sowie auf die Datenquelle bzw. Senke GSP wird im Abschnitt 5.2.2 eingegangen.

Da hier die Adreßgenerierung im System nicht beschrieben wird, sind in Bild (5.2-1) keine Indexregister zu finden. Sie können an den Multiplexer M2 angeschlossen werden und sind so mittels des Mikroprozessors modifizierbar. Werden hierzu zwei Eingänge von M2 verwendet, haben alle Multiplexer 2^n , $n = 1, 2, 3$ Eingänge, und sind somit durch gängige integrierte Schaltkreise redundanzfrei realisierbar.

5.2.1.3 Realisierung

Im folgenden wird kurz auf Fragen des Einsatzes von Pipelineregistern, sowie auf den Hardwareaufwand eingegangen.

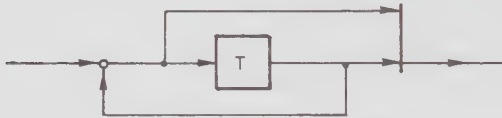
a) Einsatz von Pipelineregistern

Gemäß der Beschreibung von Abschnitt 5.1 können zur Erzielung von hohen Datendurchsatzraten zwischen jeweils zwei aufeinanderfolgende Logikeinheiten Pipelineregister angeordnet werden.

Da die Zugriffszeit eines Speichers wesentlich höher als die Verzögerungszeit eines in gleicher Technologie hergestellten Registers ist, sind die zeitkritischen Wege in Bild (5.2-1) sicherlich all diejenigen, in denen die Speichereinheiten SP1 und SP2 die Datenquellen bzw. -senken darstellen. Ein weiterer kritischer Weg stellt die ALU-Einheit des Mikroprozessors dar. Ausgehend von der - realistischen - Annahme, daß die Rechenzeit des Multiplizierers und Akkumulations-addierers in der gleichen Größenordnung wie die Taktzeit des

Prozessors liegt, werden deshalb in Bild (5.2-1) die Pipelineregister direkt an die Dateneingänge des Multiplizierers bzw. Mikroprozessors gelegt, so daß die Zugriffszeiten der Speicher und die Laufzeiten durch die Multiplexer nicht in die Rechenzeiten eingehen. Die Pipelineregister in Bild (5.2-1) sind mit P bezeichnet. Auch in der Rückführung des Ausgangssignals des Mikroprozessors zu den Multiplexern M1 und M2 befindet sich ein Register, damit sich auch beim Datentransport in die Speicher die Laufzeiten nicht aufaddieren.

In Bild (3.2-33) ist für den Akkumulator ein System vorgesehen, bei dem die Ausgangsdaten des Akkumulationsaddierers das Register umgehen können:



Diese Schaltung ist in Bild (5.2-1) nicht zu finden, weil die Struktur des Akkumulators durch den integrierten Baustein fixiert ist. Durch Einsatz eines Pipelineregisters kann sie jedoch auch mit der vorgegebenen Akkumulatorstruktur realisiert werden. Man erhält dann:



Diese Konfiguration ist identisch mit nachfolgender Schaltung, die in Bild (5.2-1) angegeben ist.



Zu den Pipelineregistern soll hier noch eine Anmerkung gemacht werden. Eine Steigerung der Leistungsfähigkeit des Systems kann erreicht werden, wenn die Pipelineregister auch als Datenspeicher verwendbar sind. Dies wird möglich, wenn sie nicht mit dem Systemtakt direkt, sondern über Gatterbausteine verbunden sind, die in Abhängigkeit von Mikrooperationsbits den Takt ein- bzw. ausschalten.

b) Bauteileaufwand

Es soll hier der Bauteileaufwand für das Minimalsystem sowie eine Abschätzung für das erweiterte System angegeben werden.

I) Minimalsystem

Das Minimalsystem wurde als 12-Bit Rechenwerk realisiert und mit einer einfachen Mikroprogrammsteuerung ausgestattet [31]. Die beiden Speicher SP1 und SP2 haben dabei eine Kapazität von jeweils 256 Worten. Für die Zahl der benötigten Bausteine und Mikrooperationsbits ergeben sich folgende Werte:

	Bausteine	μ -Operationsbits
Gatter, Multiplexer	30	12
Register, Zähler	27	6
Arbeitsspeicher	6	2
Mikroprogrammspeicher	5	9
akkumulierender Multiplizierer	1	4
Sonstige	-	6
Gesamtaufwand	69	39

Der akkumulierende Multiplizierer benötigt hierbei die Stellfläche von ca. 8 Bausteinen der Standardgröße. Das gesamte System läßt sich auf einer Doppeleuropakarte der Größe $160 \times 235 \text{ mm}^2$ unterbringen. Es wurde eine Zykluszeit von 100 nsec

erreicht, in der auch Speicherzugriffe möglich sind. Zur Produktbildung mit Akkumulation sind zwei Maschinenzyklen notwendig. Anwendungsbeispiele und Rechenzeitangaben sind in Abschnitt 5.2.3 angegeben.

II) Erweitertes System

Für das Erweiterte System nach Bild (5.2-1) soll hier eine Schätzung des Bauteileaufwandes gegeben werden, wobei eine Mikro- bzw. Makroprogrammsteuerung außer Betracht bleibt.

Es wird ausgegangen von einer Maschinenzkluszeit von 100 nsec und entsprechend schnellen Speichern sowie einer Rechenzeit von 200 nsec für den akkumulierenden Multiplizierer und den Mikroprozessor. Weiterhin sind eine Datenwortbreite von 16 Bit und eine Speicherkapazität von 2 mal 4K Worten zugrundegelegt. Es ergibt sich folgende Abschätzung:

	Bausteine	Mikrooperationsbit
Gatter, Multiplexer	71	13
Register	16	8
Arbeitsspeicher	32	2
Akkumulierender Multiplizierer	1	4
Mikroprozessor (2903)	4	26
Sonstige	-	4
Gesamtaufwand	124	57

Der akkumulierende Multiplizierer benötigt ca. 8 Stellplätze der Größe eines Standardbausteines, die vier Mikroprozessorelemente (zur Kaskadierung für 16 Bit) benötigen ca. 15 Plätze. Die Mikrooperationsbits beziehen sich auf ein horizontales Mikrowort, das zur Vermeidung von Redundanzen noch codiert werden kann.

Das Gesamtsystem läßt sich demnach auf weniger als zwei Doppelpackarten realisieren, wenn die Packungsdichte hinreichend hoch ist. Dies ist bei der Wrap-Technik gegeben.

Es sei lediglich erwähnt, daß sich eine Programmsteuerung ebenfalls auf einer Doppelpackarte implementieren läßt.

5.2.2 Kaskadierung der Teilprozessoren

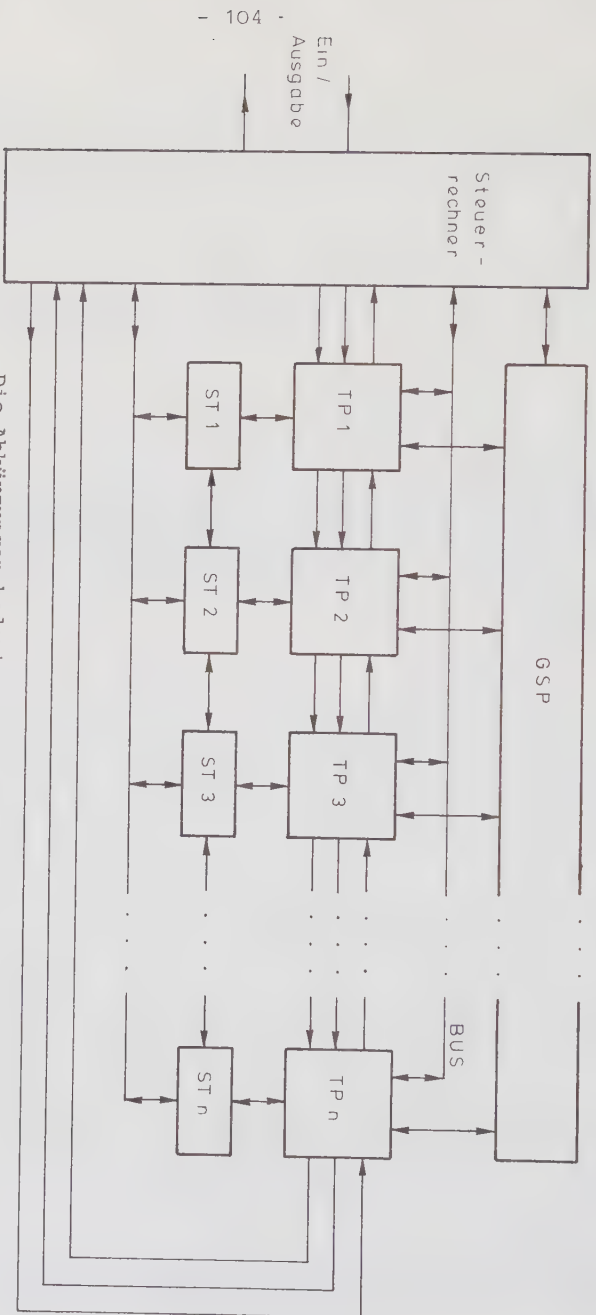
Das Prinzip der Kaskadierung von Teilprozessoren wurde bereits im Kapitel 3.1 beschrieben. Es soll nun die Struktur eines zellularen Rechnersystems angegeben werden, bei dem die beschriebenen Teilprozessoren im Multiprozessorbetrieb die gemeinsame Bearbeitung eines Algorithmus vornehmen können. Bild (5.2-2) zeigt die Architektur. Im folgenden wird die Funktion der einzelnen Systemkomponenten kurz skizziert.

5.2.2.1 Teilprozessoren

Als Teilprozessoren werden Systeme nach Bild (5.2-1) eingesetzt. Der Datenaustausch benachbarter Prozessoren wurde ausführlich im Kapitel 3 behandelt. Die dort geforderten Datenwege werden von der Architektur nach Bild (5.2-2) bereitgestellt.

5.2.2.2 Steuerwerke

Die Steuerwerke beinhalten im wesentlichen die Mikroprogrammsteuerungen der Prozessoren. Die Steuerleitungen eines einem Teilprozessor zugeordneten Steuerwerkes dienen der Übermittlung der Mikrooperationsbits und Adressen für die Speicher. Von den Teilprozessoren werden in umgekehrter Richtung Statusinformationen an die Steuerwerke gesendet, um ihnen die Ausführung bedingter Programmsprünge zu ermöglichen. Die Steuerwerke benachbarter Teilprozessoren sind gekoppelt. Dadurch lassen sich parallel zu den Daten in der Teilprozessorkaskade auch Steuerinformationen weitergeben. Dies ist z.B. notwendig, wenn die Rechenzeiten für die Bereitstellung der Ausgangsdaten in den einzelnen Teilprozessoren unterschiedlich sind, und sich deshalb benachbarte Systeme z.B. nach den in Abschnitt 4.2 beschriebenen Prinzipien synchronisieren. Nach [37] eignet sich ein solches System dann auch zur Realisierung umfangreicher experimenteller Systeme, bei denen Aufgaben der Signalerzeugung, -verarbeitung, und -auswertung in verschiedenen, nach Bild (5.2-2) gekoppelten Teilprozessoren realisiert werden.



Die Abkürzungen bedeuten:

GSP = Gemeinsamer Speicher

TPn = n-ter Teilprozessor

STn = n-tes Steuerwerk

BUS : Einzel- oder Mehrfachbus

Bild (5.2-2)

Architektur eines zellularen Multiprozessorsystems. Die Adreß- und Steuerwege von den Steuerwerken zum Gemeinsamen Speicher sind der Übersichtlichkeit wegen nicht eingezeichnet.

5.2.2.3 Gemeinsamer Speicher

Die in den Teilprozessoren nach Bild (5.2-1) enthaltenen lokalen Speicher dienen in erster Linie der Bereitstellung von Konstanten und der Zwischenspeicherung von Ergebnissen. Der allen Teilprozessoren zugängliche gemeinsame Speicher (GSP) erfüllt zwei Aufgaben:

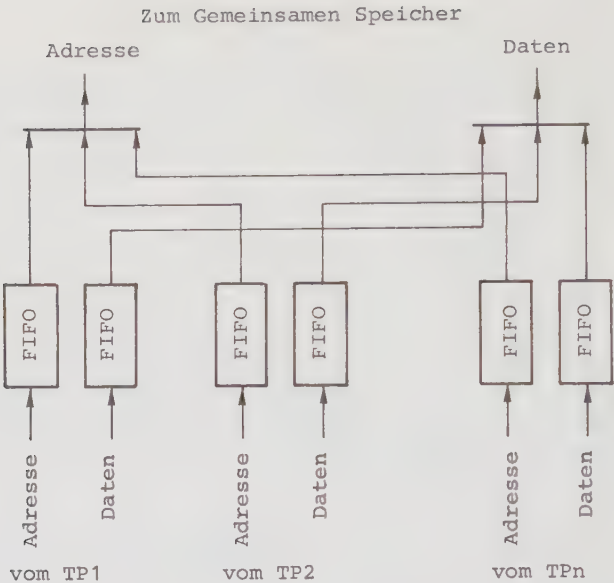
- er dient als Hintergrundspeicher, wenn die Kapazität der lokalen Speicher nicht ausreicht, und
- er bietet die Möglichkeit eines Datenaustausches auch nicht direkt benachbarter Prozessoren.

Der letzte Punkt ist insbesondere für die effektive Realisierung von Algorithmen wichtig, die sich nicht wie die in Kapitel 3 angegebenen in voneinander unabhängige Teilaufgaben zergliedern lassen.

Der Datenverkehr der Teilprozessoren mit dem gemeinsamen Speicher ist zeitkritisch, weil eine gleichzeitige Speicheranforderung mehrerer Prozessoren möglich ist. Dabei müssen zwei Fälle unterschieden werden:

Lesezyklus: Ein Teilprozessor fordert ein Datenwort an und steht damit in Konkurrenz zu seinen Nachbarn. Er muß deshalb auf eine Zuteilung des Speichers warten (Strategien der Prioritätszuteilung werden hier nicht betrachtet).

Schreibzyklus: Bei einem Schreibzyklus ist ein einzelner Teilprozessor unabhängig von der Reaktion des Speichers, er kann die Operation dem Speicher zur selbständigen Ausführung überlassen. Deshalb ist es möglich, die Anforderung eines Schreibzyklus in einem "First in First out"-Speicher abzulegen:



So lange die Kapazität der Pufferspeicher hinreichend groß ist, beeinflusst die Zahl der Anforderungen oder auch die Zugriffszeit des Speichers nicht die Rechengeschwindigkeit der Teilprozessoren, sondern nur die des Gesamtsystems.

Um eine vom Gemeinsamen Speicher unabhängige Möglichkeit zum Datenaustausch bereitzustellen, sind sämtliche Prozessoren zusätzlich noch durch einen (oder mehrere) BUS(SE) (siehe Bild (5.2-2)) verbunden.

5.2.2.4 Steuerrechner

Der Steuerrechner übernimmt im wesentlichen die Kontrolle des Gesamtsystems und seine Verbindung mit peripheren Einheiten. Seine Aufgaben seien hier lediglich durch einige Stichpunkte beschrieben. Er ermöglicht

- die Verbindung mit peripheren Geräten wie Drucker, Stanzer, Plattenspeicher, A/D- und D/A-Wandlern,
- die Kommunikation mit einer Steuerkonsole,

- das Programmladen sowie den Programmstart
- und schließlich die Bereitstellung von Programmierungshilfen wie Assembler- und Fehlersuchprogramme.

Für diese Aufgaben ist als Steuerrechner ein universeller Prozeßrechner geeignet, da sie nicht zeitkritisch sind. Eine Ausnahme hiervon kann die analoge Ein-/Ausgabe sein, die jedoch durch direkte Verbindung der Ein- bzw. Ausgangskanäle der Prozessorkaskade der Geschwindigkeit der Teilprozessoren anpaßbar ist.

5.2.3 Anwendungsbeispiele

Es wurde bislang lediglich das in Abschnitt 5.2.1.1 und 5.2.1.3 beschriebene Minimalsystem realisiert. Es sollen hier ausschließlich bereits erreichte Rechenleistungen dargestellt werden, die folgende Tabelle gibt deshalb nur Werte für das Minimalsystem an. Sie sind aus [31] entnommen.

	nichtrekurs. Filter	rekursives Filter	Kreuz-/Autokorr.
Filtergrad n	n = 1...255	n = 1...51	-
Korrelationslänge, Korrelationswerte	-	-	2N+1 = 255
Länge des Mikro- Programmes	11 µ-Progr.- Worte	22 µ-Progr.- Worte	22 µ-Progr.- Worte
Auslastung des Multiplizierers	29% für n=1 99% für n=255	67% für n=1 99% für n=51	98%
Rechenzeit	$(5+2n) \cdot 0,1 \mu\text{sec}$	$(6+12\frac{n}{2}) \cdot 0,1 \mu\text{sec}$	6656 µsec

Im Fall des rekursiven Filters wird dabei eine stabile Überlaufkennlinie nach Abschnitt 3.2.2.2, und Zufallsrunden für die Wortlängenverkürzung eingesetzt.

Die Berechnung des Schätzwertes der Auto- bzw. Kreuzkorrelationsfunktion wird gemäß der Beziehung (3.2-7) durchgeführt.

Die erreichten Rechenzeiten und die Auslastung des Multiplizierers bestätigen die Eignung des Systems für die ausgewählten Algorithmen der Signalverarbeitung.

5.3 Universeller digitaler Signalprozessor

Es wird nun ein universeller digitaler Signalprozessor (DSP) beschrieben, der im Rahmen eines von der Deutschen Forschungsgemeinschaft geförderten Forschungsvorhabens entworfen und aufgebaut wurde.

Das System wurde speziell für die Realisierung von Filteralgorithmen und Vektor- bzw. Matrizenoperationen für die zweidimensionale Signalverarbeitung ausgelegt. Ein wesentliches Ziel bei seiner Konzeption war die Bereitstellung einer hohen Rechenleistung für die erwähnten Aufgaben. Dabei sollte dem Benutzer die Möglichkeit zur Programmierung auf Maschinenbefehlsebene gegeben werden, so daß die Notwendigkeit zur Mikroprogrammierung entfällt. Um deshalb keine Geschwindigkeitseinbußen hinnehmen zu müssen, werden sämtliche Befehle durch direkte Hardwaresteuerung in einem bzw. zwei Maschinenzyklen ausgeführt. Hieraus resultiert ein vergleichsweise hoher Schaltungsaufwand.

Die Rechenleistung des im vorangegangenen Abschnitt beschriebenen zellularen Multiprozessorsystems läßt sich durch die Menge der eingesetzten Teilprozessoren in weiten Grenzen variieren. Mit zunehmender Zahl von Prozessoren nehmen jedoch die Probleme bei der Programmierung des Systems zu, weil die Programme auf die zeitlichen Zusammenhänge der Algorithmenberechnung in unterschiedlichen Rechenwerken Rücksicht nehmen müssen. (Dies gilt jedoch nur stark abgeschwächt für eine Kaskadierung nach den Prinzipien von Kapitel 3).

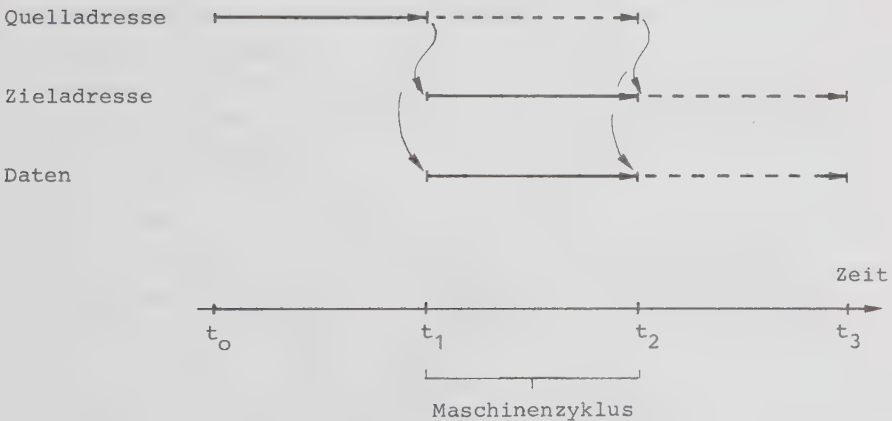
Bei dem hier beschriebenen Konzept wird deshalb die Zahl der Rechenwerke reduziert auf einen Teilprozessor mit erhöhter innerer Rechenleistung, wie er im Abschnitt 3.2.7.2 beschrieben wurde. Dieses Rechenwerk wird integriert in einen leistungsfähigen universellen Rechner, bei dessen Entwurf die speziellen Erfordernisse dieser "Arithmetischen Einheit" konsequent berücksichtigt werden.

5.3.1 Systemarchitektur

Bild (5.3-1) zeigt die Systemarchitektur des Digitalen Signalprozessors. Die Zentraleinheit (CPU) bildet darin das zentrale Steuer- und Rechenwerk für das Gesamtsystem, sie bearbeitet Maschinenbefehle die im Programmspeicher abgelegt sind.

Die Zentraleinheit ist mit der Ein-/Ausgabeeinheit, der Arithmetischen Einheit die den Teilprozessor realisiert, sowie den Speichermoduln und dem Linkregister über ein Bussystem verbunden. Dadurch entsteht der Vorteil, daß das Gesamtsystem um zusätzliche Einheiten, die an den Bus anschließbar sind, einfach erweitert werden kann.

Das Bussystem enthält einen Steuerbus, der den angeschlossenen Einheiten die Interpretation der Daten auf den anderen Bussen erlaubt, einen Datenbus zum Datenaustausch, sowie einen Quell- adreß- und Zieladreibus. Die Verwendung zweier Adreßbusse gestattet das gleichzeitige Lesen und Schreiben in voneinander unabhängigen Einheiten, wie nachstehende Skizze zeigt.



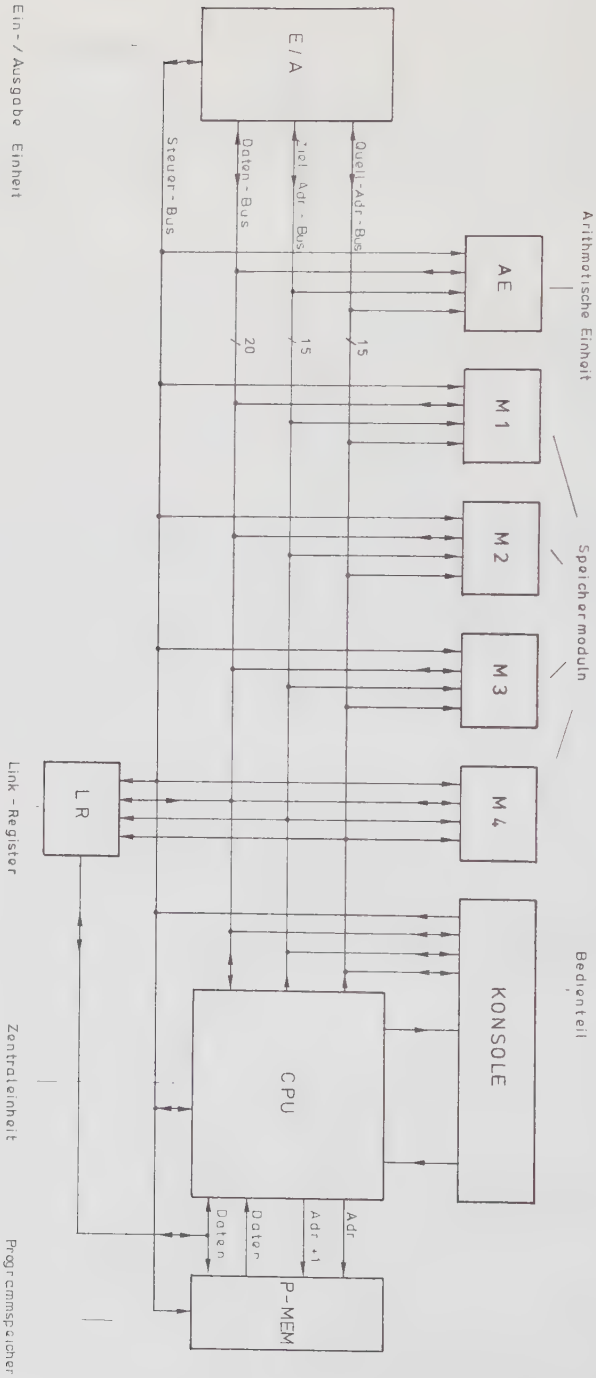


Bild (5.3-1)

Systemarchitektur des DSP

Zur Zeit t_0 wird auf dem Quelladreibus eine Leseadresse aktiv, weshalb die angesprochene Datenquelle bis zum Zeitpunkt t_1 das geforderte Datenwort in ein ihr zugeordnetes Pipelineregister schreibt und während des Zeitintervalles (t_1, t_2) den Inhalt des Pipelineregisters über den Datenbus dem System mitteilt.

Im selben Zeitintervall stellt der Zieladreibus eine Schreibadresse zur Verfügung, so daß die adressierte Einheit die Daten des Datenbusses übernehmen kann. Während dieser Zeit kann eine andere Einheit durch den Quelladreibus zur neuerlichen Bereitstellung von Daten veranlaßt werden.

Durch den Einsatz der beiden Adreibusse läßt sich somit die Transportkapazität des Datenbusses optimal ausnutzen.

Neben dem dreifach Bussystem ist die Verwendung eines von den Speichermoduln getrennten Programmspeichers ein wesentlicher Punkt der Systemarchitektur nach Bild (5.3-1). Diese Trennung erlaubt eine überlappende Bearbeitung der Befehlshol- sowie der Befehlsausführungsphase. Dadurch ist vor allem deshalb eine entscheidende Geschwindigkeitssteigerung zu erwarten, weil die Befehlsausführungszeiten in der gleichen Größenordnung liegen wie die zum Holen der Befehle notwendige Zeit.

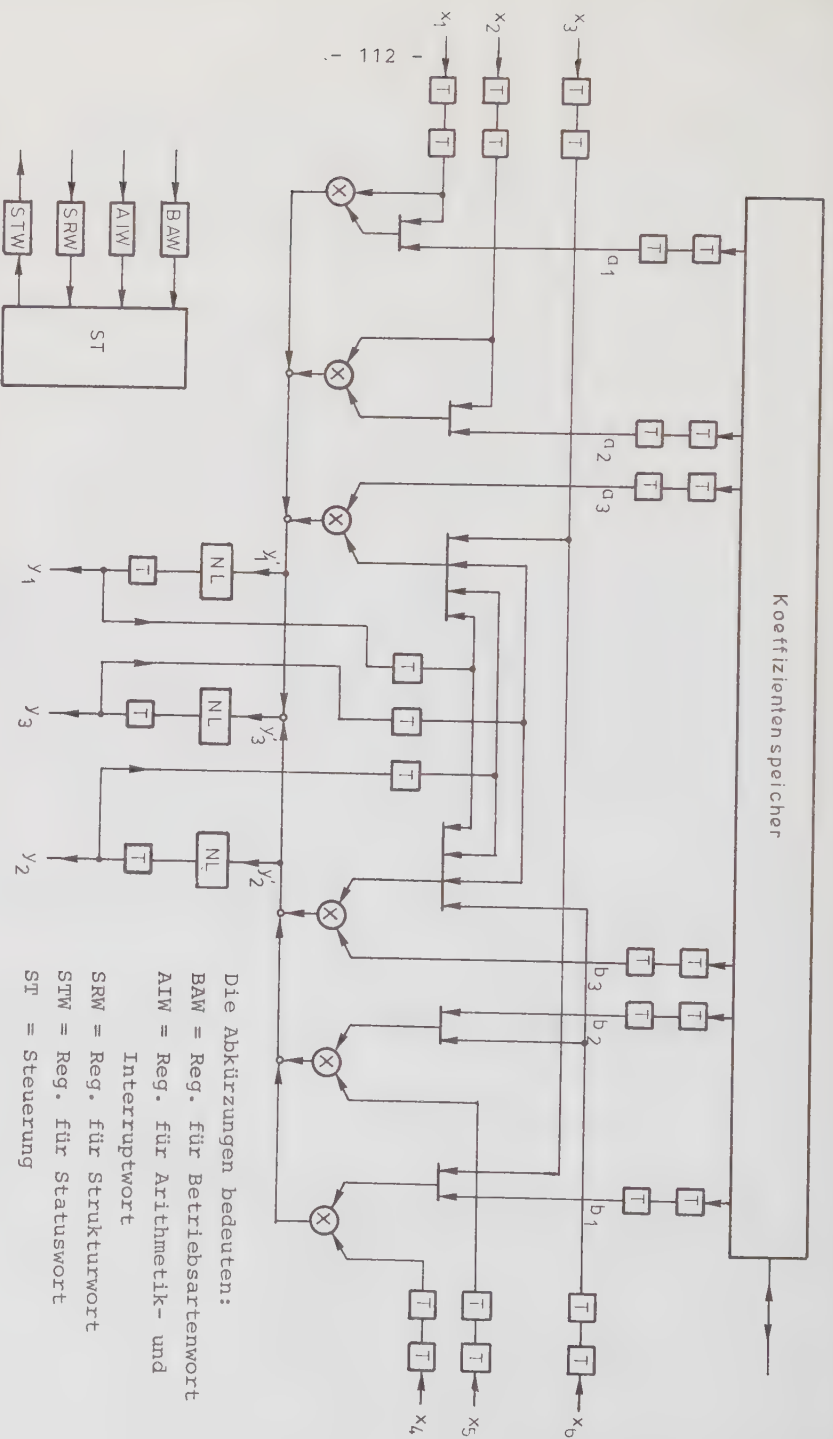
Der Programmspeicher ist über das "Link-Register" mit dem Bussystem verbunden, so daß z.B. Programme über die Eingabeeinheit geladen werden können.

Im folgenden werden die einzelnen Komponenten des Prozessors detailliert beschrieben.

5.3.1.1 Teilprozessorstruktur

Bild (5.3-2) zeigt die Struktur der Arithmetischen Einheit, die hier den Teilprozessor realisiert. Sie liefert drei Ausgangswerte pro Rechenzyklus, die gemäß

$$y_1(k) = \alpha_1 \begin{bmatrix} a_1 \\ x_1(k) \end{bmatrix} x_1(k) + \alpha_2 \begin{bmatrix} a_2 \\ x_2(k) \end{bmatrix} x_2(k) + \alpha_3 a_3 \begin{bmatrix} x_3(k) \\ y_v(k-1), v=1,2,3 \end{bmatrix}$$



112

Bild (5.3-2)

Struktur der Arithmetischen Einheit. Sämtliche Ein- und Ausgänge führen zum Datenbus. Die Addierwerke können auch Subtrahieren, die Vorzeichenfaktoren a_1 , b_1 sind nicht eingezeichnet

- Die Abkürzungen bedeuten:
- BAW = Reg. für Betriebsartenwort
 - AIW = Reg. für Arithmetik- und Interruptwort
 - SRW = Reg. für Strukturwort
 - STW = Reg. für Statuswort
 - ST = Steuerung

$$y_2(k) = \beta_1 \begin{bmatrix} b_1 \\ x_3(k) \end{bmatrix} x_4(k) + \beta_2 \begin{bmatrix} b_2 \\ x_6(k) \end{bmatrix} x_5(k) + \beta_3 b_3 \begin{bmatrix} x_6(k) \\ y_v(k-1), v=1,2,3 \end{bmatrix}$$

$$y_3(k) = y_1(k) + y_2(k)$$

mit	$\alpha_v, \beta_v \in \{+1, -1\}$	$v = 1, 2, 3$
	$a_v, b_v =$ Koeffizienten	$v = 1, 2, 3$
	$x_v =$ Eingangsvariable	$v = 1, 2, 3, 4, 5, 6$
	$y_v =$ Ausgangsvariable	$v = 1, 2, 3$

errechnet werden [38]. Die Klammern deuten dabei alternative Möglichkeiten an.

Die Eingangsvariablen x_v werden über den Datenbus in zwei kaskadierte Pipelineregister des Rechenwerkes transportiert. Die Verwendung von zwei Registern erlaubt ein erneutes Laden von Eingangsdaten noch während die Arithmetische Einheit arbeitet.

In der Arithmetischen Einheit werden die 6 Produkte sowie die 5 Summen mittels der sog. "Verteilten Arithmetik" realisiert. Die Einzelheiten dieses Schaltungsprinzips werden hier nicht behandelt, es sei lediglich auf [39] verwiesen.

Für die Darstellung der Koeffizienten und Variablen wird ausnahmslos das 2'er Komplement nach Gleichung (3.2-25) verwendet, wobei die Wortlänge 20 Bit beträgt.

Für die Ausführung der durch die oben angegebenen Gleichungen beschriebenen arithmetischen Operationen müssen für einen Rechenzyklus bis zu 6 Eingangsvariable und bis zu 3 Ergebnisdatenworte über das Bussystem nach Bild (5.3-1) transportiert werden. Hinzu kommt der Transport eines Steuerwortes in das Betriebsartenregister, der den Rechenzyklus auslöst. Die realisierte Arithmetische Einheit benötigt eine Rechenzeit von 1950 μ sec, so daß in dieser Zeit bis zu 10 Datentransfers von der Zentraleinheit ausgeführt werden müssen. Rechnet man 3 weitere Transfers z.B. zur Bedienung der Ein-/Ausgabeeinheit hinzu, resultiert für die Zentraleinheit die Forderung nach einem Datentransfer inklusive Befehlsholphase und Adreßrechnung in 150 nsec. Als Vergleich sei erwähnt, daß der Prozeßrechner

PDP 11/45 für einen solchen Befehl ca. 4 μ sec benötigt ("MOV" Speicher nach Speicher, absolute Adressierung). Der gesamte Entwurf der Zentraleinheit des DSP orientiert sich an diese Forderung.

Im folgenden wird die Funktion der einzelnen Schaltungsblöcke in Bild (5.3-2) erläutert.

a) Koeffizientenspeicher

Der Koeffizientenspeicher hat eine Kapazität von 2K Worten, er kann demnach bis zu 341 Koeffizientensätze für das Rechenwerk bereithalten. Ein detailliertes Blockschaltbild ist in Bild (5.3-3) gegeben.

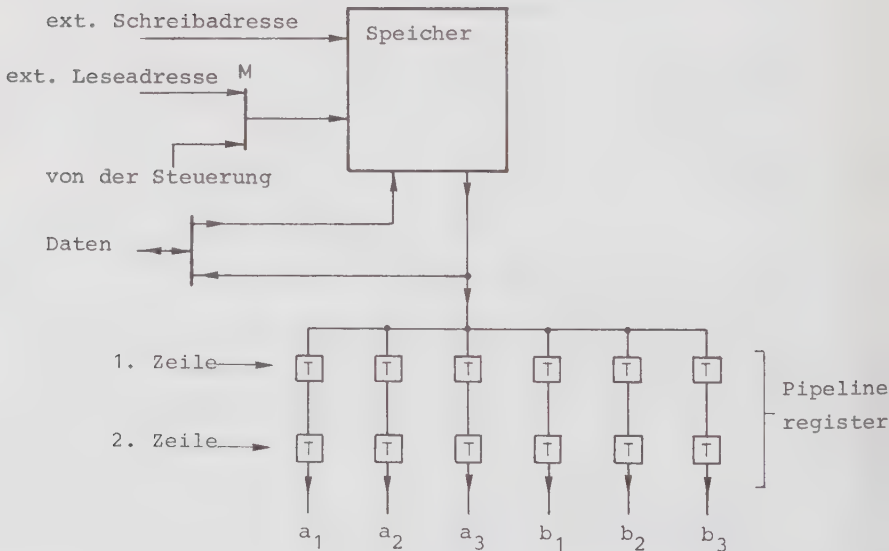


Bild (5.3-3) Aufbau des Koeffizientenspeichers der Arithmetischen Einheit

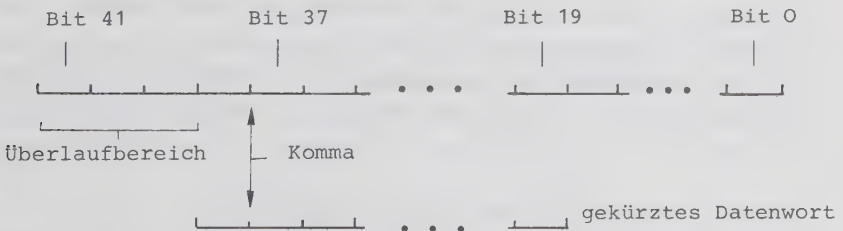
Der Speicher kann durch Bereitstellung einer externen Leseadresse durch den Quelladreßbus (siehe Bild 5.3-1) Daten an das Rechenwerk außerhalb der Arithmetischen Einheit abgeben, bzw. externe Daten in die durch die externe Schreibadresse vorgegebene Speicherzelle einschreiben. Entscheidend für die Funktion der Arithmetischen Einheit ist jedoch die Möglichkeit, daß die Steuerung ST in Bild (5.3-2) eine eigene Leseadresse über den Multiplexer M zum Speicher in Bild (5.3-3) leiten kann. Somit sind die Pipelineregister zeitlich sequentiell mit Daten aus dem Speicher beschreibbar.

Die Kaskadierung zweier Pipelineregister für jeden Koeffizienten ermöglicht auch hier das Einschreiben von Datenworten in die erste Zeile der Register, während die arithmetische Einheit den Inhalt der zweiten Zeile als Koeffizienten für einen Rechenzyklus verwendet.

b) Nichtlinearitäten

In Bild (5.3-2) realisieren die Blöcke "NL" die Nichtlinearitäten, die in Abschnitt 3.2.2.2 beschrieben wurden. Wegen der Zahlendarstellung durch das 2'er-Komplement mit einer Wortlänge von 20 Bit entstehen Produkte mit einer Wortlänge von 39 Bit. Da in der Arithmetischen Einheit bis zu 6 solcher Produkte aufsummiert werden, sind weitere 3 Bit zur Darstellung des Zahlenbereichs $-6 \leq y_3' < +6$ notwendig. (Für die Ausgangsgrößen y_1' und y_2' sind 2 Bit ausreichend, der Einheitlichkeit wegen werden auch hier 3 Bit vorgesehen).

Es ergibt sich somit für die Darstellung der Ausgangswerte folgende Binärstellenzuordnung:



Diese 42-stellige 2'er-Komplementzahl muß durch nichtlineare Operationen auf eine Länge von 20 Bit gekürzt werden.

I) Verminderung der Nachkommastellen

Die Zahl der Nachkommastellen muß 19 betragen. Die Nicht-linearitäten realisieren 4 Methoden der Wortlängenverkürzung:

1. 2'er-Komplement-Abschneiden
2. 2'er-Komplement-Runden
3. Betrags-Abschneiden
4. Zufallsrunden

Ihre Wirkung in digitalen Systemen zur Signalverarbeitung wird z.B. in [13, 14] beschrieben.

II) Überlaufbehandlung

Die 3 Bit des Überlaufbereiches in obiger Zahlendarstellung für die Ergebnisworte y' müssen zu einem Vorzeichenbit reduziert werden. Dafür werden 4 Möglichkeiten zur Verfügung gestellt:

1. Abschneiden von Bit 39...41 (keine Überlaufbehandlung)
2. Dreieckskennlinie
3. Sättigungskennlinie
4. Nullsetzen sämtlicher Binärstellen und Abschneiden von Bit 39...41 .

Es sei noch erwähnt, daß die Überlaufbehandlung für die in ihren Nachkommastellen verkürzte Zahl eingesetzt wird, da die Rundungsoperation einen Überlauf erzeugen kann.

c) Steuerregister

In Bild (5.3-2) sind mit den Abkürzungen "BAW", "AIW", "SRW" und "STW" Steuerregister eingezeichnet, die von der Zentraleinheit außerhalb der Arithmetischen Einheit beschrieben bzw. gelesen werden können. Ihre Funktionen werden nachfolgend erläutert.

I) Register für das Betriebsartenwort (BAW)

Das Beschreiben dieses Registers löst einen Rechenzyklus der Arithmetischen Einheit aus. Es wird im wesentlichen geladen mit der Anfangsadresse des im nächsten Zyklus gewünschten

Koeffizienten. Die Produktsummen werden gebildet mit den Koeffizienten, die sich in der zweiten Zeile der Pipelineregister in Bild (5.3-3) befinden, da mit dem Start der Arithmetischen Einheit der Inhalt der ersten Zeile in die zweite Zeile transportiert wird. Wenn beim Beschreiben des Registers für das Betriebsartenwort keine Anfangsadresse mitgeteilt wird, so werden die auf den letzten Koeffizientenspeicherzugriff folgenden 6 Koeffizienten in die erste Zeile der Pipelineregister geladen. Somit ist gewährleistet, daß das Auslesen des Koeffizientenspeichers keine zusätzliche Rechenzeit benötigt, da es während dem Rechenzyklus der Arithmetischen Einheit realisiert wird.

II) Register für das Arithmetik- und Interruptwort (AIW)

Dieses Register enthält das Codewort, mit dem die Arithmetik des Rechenwerkes sowie die Interruptquellen beschrieben werden. Es definiert die Nichtlinearitäten für jede der drei Ausgangsvariablen y_1 , y_2 und y_3 , und stellt eine Interruptmaske zur Verfügung, mit deren Hilfe Programmunterbrechungen ermöglicht werden, wenn z.B. im Überlaufbereich der Binärdarstellungen von y_1' , y_2' oder y_3' nicht sämtliche Bit übereinstimmen, also ein Überlauf stattgefunden hat.

III) Register für das Strukturwort (SRW)

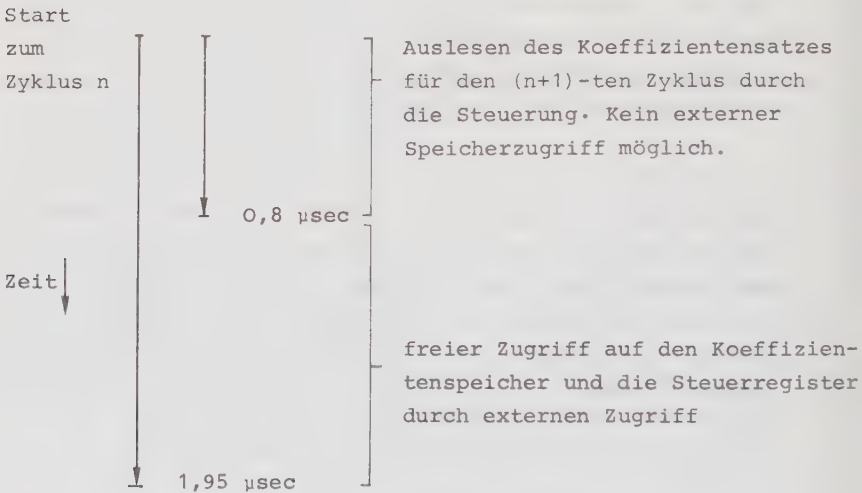
Die einzelnen Binärstellen dieses Registers sind direkt den Multiplexern der Arithmetischen Einheit zugeordnet, die die Datenwege schalten. Der Inhalt des Registers enthält somit die Datenpfadstruktur des Rechenwerkes.

d) Register für das Statuswort (STW)

Das Register für das Statuswort ist von der Zentraleinheit über den Datenbus lesbar. Es enthält Informationen z.B. über die Vorzeichen der Ausgangsvariablen y_1 , y_2 und y_3 , oder gibt Auskunft, wo sich ein Überlauf ereignet hat.

e) Steuerung (ST)

Der Schaltungsteil ST der Arithmetischen Einheit dient in erster Linie der Interpretation des Inhaltes der Steuerregister und der Bereitstellung der sequentiellen Steuer-signale für die Komponenten des Rechenwerkes. Nachfolgend wird der zeitliche Arbeitsablauf in der Arithmetischen Einheit angegeben.



Das Beschreiben der Eingangsvariablenregister ist während dem gesamten Rechenzyklus der Arithmetischen Einheit möglich.

5.3.1.2 Zentraleinheit

a) Aufgaben und Struktur

Aufgabe der Zentraleinheit ist die Ausführung der im Programmspeicher befindlichen Maschinenbefehle sowie die Instruktionen, die der Benutzer dem System manuell über die Konsole mitteilt.

Ein stark vereinfachtes Strukturdiagramm zeigt Bild (5.3-4). Es gibt die Aufteilung in einen Datenverknüpfungsteil und einen Steuerteil wieder.

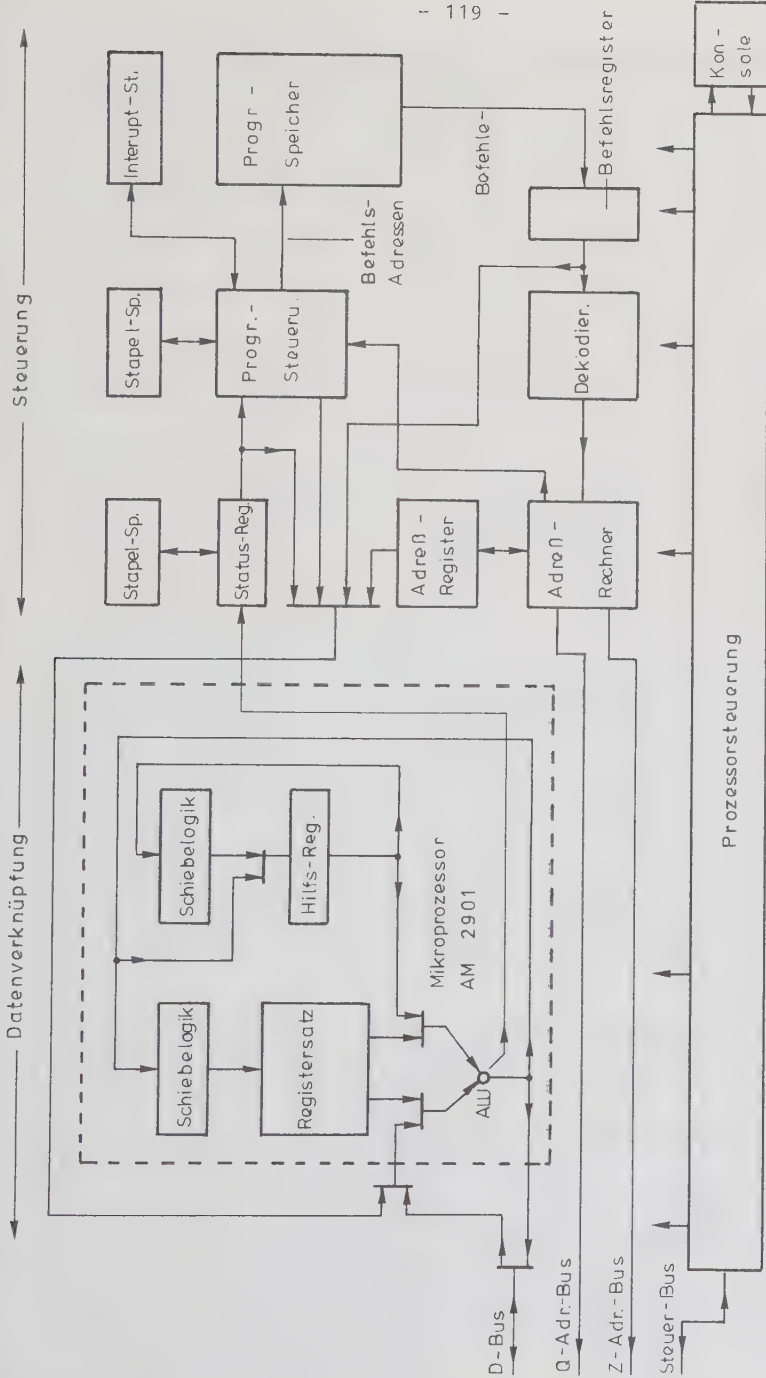


Bild (5.3-4) Stark vereinfachte Struktur der Zentraleinheit des DSP

Der Datenverknüpfungsteil wird im wesentlichen realisiert durch einen in Bipolartechnik aufgebauten mikroprogrammierbaren Mikroprozessor des Typs AM 2901. Die wichtigsten Datenpfade dieses Bausteins sind in Bild (5.3-4) eingezeichnet. Es werden fünf kaskadierte Mikroprozessorelemente verwendet, so daß sich eine Datenwortbreite von 20 Bit ergibt.

Die Mikroprozessorelemente stellen 16 Allzweckregister sowie ein Hilfsregister als lokale Speicher zur Verfügung. Sämtliche dieser Datenquellen können die Operanden für das Verknüpfungnetzwerk (ALU) liefern. Als externe Datenquelle kann entweder der Datenbus oder weitere Register (z.B. Adreßregister, Statusregister) der Zentraleinheit fungieren. Ergebnisdaten können über die Verschiebenetzwerke des Mikroprozessors in eines seiner Register transportiert werden. Diese Operationen werden durch Maschinenbefehle ausgelöst.

Der Steuerteil der Zentraleinheit realisiert das ordnungsgemäße Auslesen der Maschinenbefehle aus dem Programmspeicher mittels der Programmsteuerung, sowie die Bereitstellung von Quell- und Zieladressen durch den Adreßrechner unter Verwendung von jeweils 7 Index-, Offset- und Moduloregistern. Die detaillierte Beschreibung dieser Einheiten ist Gegenstand der folgenden Abschnitte.

Die Zentraleinheit realisiert ca. 280 verschiedene Maschinenbefehle (inklusive aller Adressierungsmodi), die sich in folgende Befehlsgruppen aufteilen lassen.

1. Arithmetische Befehle: z.B. Addition zweier Registerinhalte, Teilschritt für die Division zweier Registerinhalte
2. Logische Befehle: z.B. bitweise UND-Verknüpfung zweier Registerinhalte, Multiplikation/Division mit 2
3. Datentransportbefehle: z.B. (Register nach Register)-, (Speicher nach Register)-Transport

- 4. Steuerbefehle : z.B. Interruptsteuerung, Sprungbefehle
- 5. Adreßregisterbefehle : z.B. Inkrementieren eines Indexregisters, Laden eines Moduloregisters.

Auf eine ausführliche Darstellung des Befehlssatzes wird hier verzichtet, es sei lediglich auf[40] verwiesen.

Im folgenden Abschnitt werden die Teilschritte beschrieben, die zur Ausführung der Befehlsgruppen 1, 2 und 3 notwendig sind.

Die Befehlsbearbeitung in der Gruppe 4 und 5 läßt sich nicht einheitlich darstellen, zudem sind z.T. sehr komplexe Teiloperationen notwendig, so daß sie aus Platzgründen hier nicht beschrieben werden.

b) Befehlsbearbeitung

Der DSP ist eine 3-Adreß-Maschine für (Speicher/Register)-Referenzbefehle, so daß z.B. ein Additionsbefehl folgendes Format hat:

ADD <Speicherzelle> , R0, R15 .

Die Aufgabe der Zentraleinheit ist hierbei neben der Ausführung der Addition noch die Bereitstellung der Speicheradresse sowie der Datentransport aus dem Speicher und dem Register 0 zum Rechenwerk (ALU), in dem die Addition durchgeführt wird. Anschließend muß das Ergebnis zum Register 15 gebracht werden.

Zur Bearbeitung eines Maschinenbefehles müssen von der Zentraleinheit demnach folgende Teilschritte T_v durchgeführt werden:

- T_1 : Befehl holen
- T_2 : Befehl interpretieren (= dekodieren)
- T_3 : evtl. Adressen errechnen
- T_4 : Befehl ausführen, z.B. Verknüpfen, Speicher lesen
- T_5 : Befehl ausführen, z.B. Verknüpfen, Speicher beschreiben

Dabei ist T_3 , T_4 und T_5 abhängig von Befehlstyp, also vom Ergebnis des Teilschrittes T_2 .

In einem konventionellen Rechenwerk werden diese Teilschritte der Befehlsausführung sequentiell bearbeitet, so daß sich für die Ausführungszeit ergibt

$$t_{Amax} = t(T1) + t(T2) + t(T3) + t(T4) + t(T5)$$

wenn $t(T_v)$ die Ausführungszeit des v -ten Teilschrittes ist. Durch den gewählten Schaltungsentwurf für den DSP läßt sich erreichen, daß die Ausführungszeiten der Teilschritte ungefähr gleich groß und in jedem Fall kleiner als eine Elementarzeit t_{Zyklus} sind. Dabei läßt sich Teilschritt 2 und 3 in einem Zyklus bearbeiten. Es gilt dann

$$t_{Amax} = 4 \cdot t_{Zyklus}$$

und es läßt sich eine 4-stufige Pipelineanordnung angeben, in der jede Pipelinestufe einen Teilschritt realisiert. Dies zeigt Bild (5.3-5). Die Befehlsdurchsatzrate wird dabei

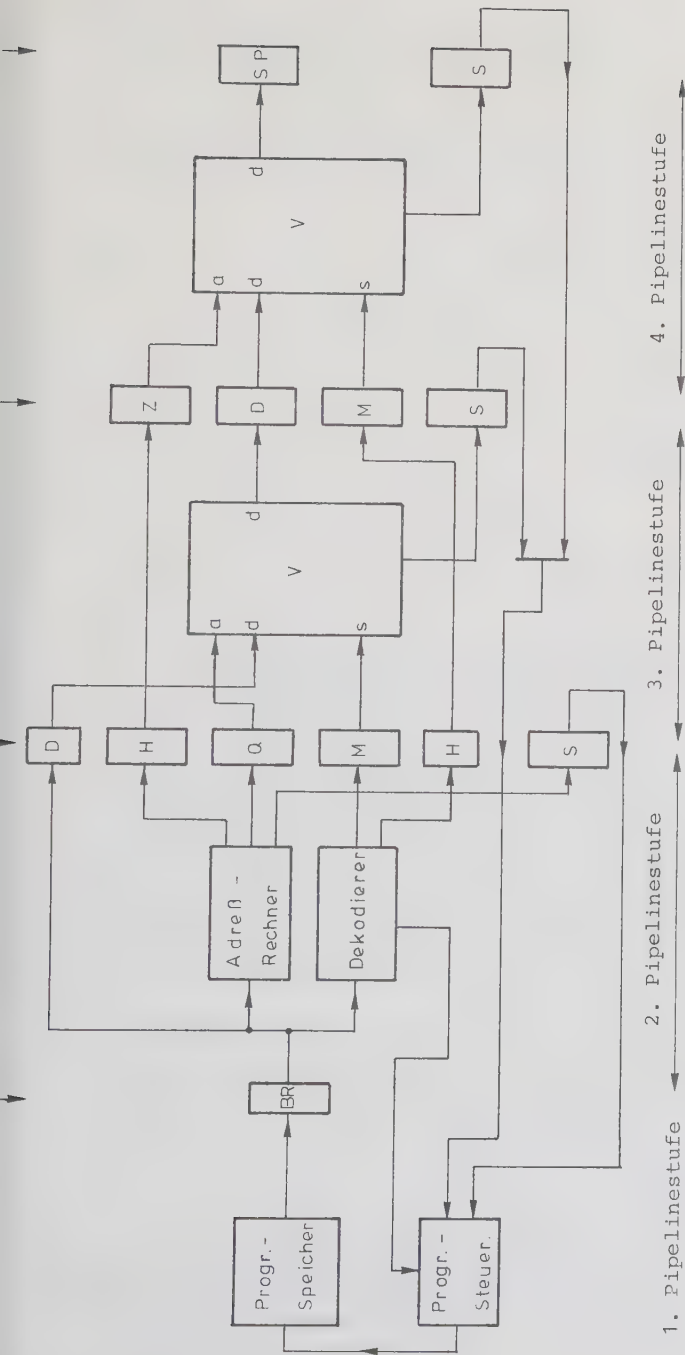
$$B = \frac{1}{t_{Zyklus}}$$

Im DSP wurde eine Elementarzeit $t_{Zyklus} = 150$ nsec erreicht, dies entspricht einer Befehlsdurchsatzrate von 6,6 Millionen Befehlen pro Sekunde. Ist für jeden Befehl in jeder Pipelinestufe eine Teiloperation notwendig, so werden maximal 26,4 Millionen Teiloperationen pro Sekunde durchgeführt.

Es wird nun der prinzipielle zeitliche Ablauf der Befehlsbearbeitung in dem System nach Bild (5.3-5) beschrieben. Dabei sind vier Fälle zu unterscheiden.

I. Normalfall

Aus der Beschreibung des Pipelineprinzipes in Abschnitt 5.1 geht hervor, daß zwischen jeweils zwei Pipelineregistern ein separates Logikelement angeordnet sein muß, da alle Elemente gleichzeitig arbeiten und deshalb ein Element nicht für zwei unterschiedliche Pipelinestufen gleichzeitig zur Verfügung steht. Wenn in der Pipelinestruktur von Bild (5.3-5) diese

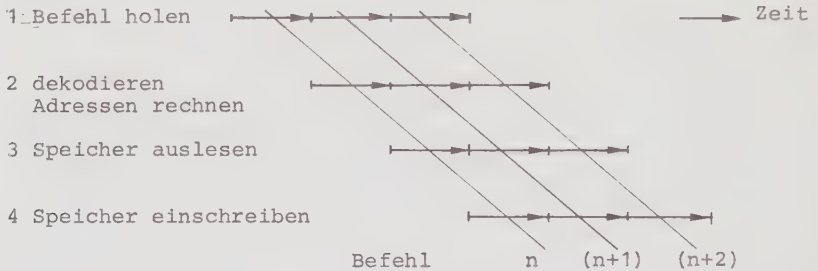


Die Abkürzungen bedeuten: BR: Befehlsregister D: Datenregister H: Hilfsregister
Q : Quelladrefregister M: Mikrobefehlsregister S: Statusregister
V : Verarbeitungslogik Z: Zieladrefregister SP: Speicher (-register)
a : Adresse d: Daten s: Steuersignale

Bild (5.3-5) Schema der Pipelinestruktur der Zentraleinheit

Forderung für die Verknüpfungslogik V in der dritten und vierten Pipelinestufe erfüllt ist, wird dies hier als der Normalfall bezeichnet. Ein Beispiel hierfür ist der Datenaustausch zwischen unterschiedlichen Speichermoduln. Es ergibt sich dann folgendes Schema der Befehlsbearbeitung:

Pipelinephase:



In der dritten Pipelinephase wird von dem Mikrooperationsregister ein Steuerwort S an das Bussystem gegeben. Dieses bewirkt, daß die angeschlossenen Datenquellen die Adresse des Quelladreibusses dekodieren. Die adressierte Quelle transportiert daraufhin ein Datenwort in das ihr zugeordnete Datenregister, dessen Inhalt sie im nächsten Pipelinezyklus über den Datenbus in das System einspeist. In der vierten Phase wird wiederum ein Steuerwort S mittels des Steuerbusses der Verarbeitungslogik (hier: Bussystem) mitgeteilt, so daß nun die durch die Zieladresse spezifizier- te Datensenke die Daten des Datenbusses in ihr Speicherregister einschreiben kann.

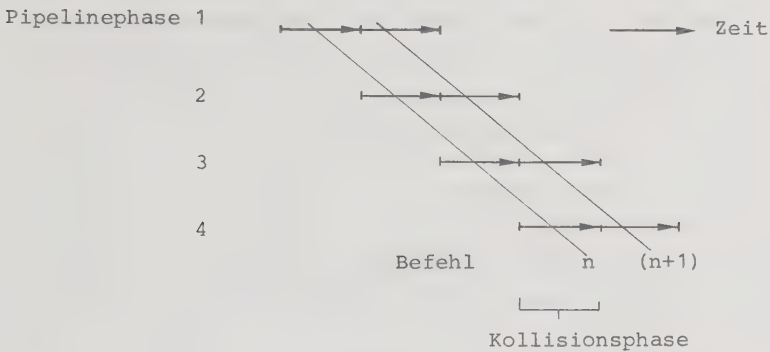
Aus dem oben angegebenen Schema geht hervor, daß diese Vorgänge störungsfrei in jeder Pipelinestufe ausgeführt werden können.

II. Kollisionsfall

IIa) Sachverhalt

Der Kollisionsfall in der Befehlsbearbeitung tritt immer dann ein, wenn in der vierten Pipelinephase des Befehls n dieselbe Verarbeitungslogik wie in der dritten Phase des Befehls $(n+1)$ angesprochen wird.

Es ergibt sich dann folgendes Schema:



In der Kollisionsphase wird eine Einheit z.B. gleichzeitig zum Lesen und Einschreiben von Daten veranlaßt. Hierzu gibt es zwei Möglichkeiten

- 1.) Die Kollision findet in einer am Bussystem angeschlossenen Einheit statt (z.B. ein Ein- und Ausgaberegister der Arithmetischen Einheit wird gleichzeitig adressiert).
- 2.) Im Befehl n wird ein Verknüpfungsergebnis der Zentraleinheit in ein Register des Registersatzes in der Phase vier eingeschrieben, gleichzeitig erfordert der Befehl (n+1) ein Auslesen eines der Register des Registersatzes in seiner Phase drei.

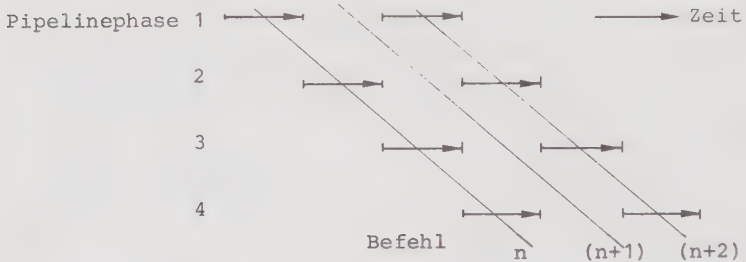
IIb) Vermeidung des Kollisionsfalls

Das Vorhandensein mehrerer Speichermoduln am Systembus erlaubt häufig eine Programmierung derart, daß der Kollisionsfall nicht eintritt. In der Befehlsfolge

TRA	Quelle 1, Ziel 1	(5.3-1)
TRA	Quelle 2, Ziel 2	

mit TRA = Transportbefehl

wird dann mit der Zieladresse 1 eine andere Einheit als mit der Quelladresse 2 adressiert. Ist dies vom Programm unvermeidbar, muß zwischen beide Transportbefehle ein Befehl (NOP = keine Operation) eingefügt werden. Dann ergibt sich folgender Phasenverlauf:



für die Befehlsfolge

n	TRA	Quelle 1, Ziel 1
n+1	NOP	
n+2	TRA	Quelle 2, Ziel 2

in der nun keine Kollision mehr möglich ist.

Programme, in denen hauptsächlich Registerreferenzbefehle vorkommen, würden bei dieser Methode der Vermeidung des Kollisionsfalles jedoch vergleichsweise viele NOP-Befehle benötigen, was einen entsprechenden Anstieg der Programm-länge und damit des Speicherplatzbedarfs zur Folge hätte.

Das Bearbeitungsprinzip dieser Befehlsklasse ist deshalb ein anderes:

- (Register nach Speicher)- bzw. (Speicher nach Register)-Typ :

Die Operanden befinden sich dabei im Speicher sowie in den Registern, z.B.:

ADD R1, R2, <Speicher>
oder ADD <Speicher>, R1, R2 .

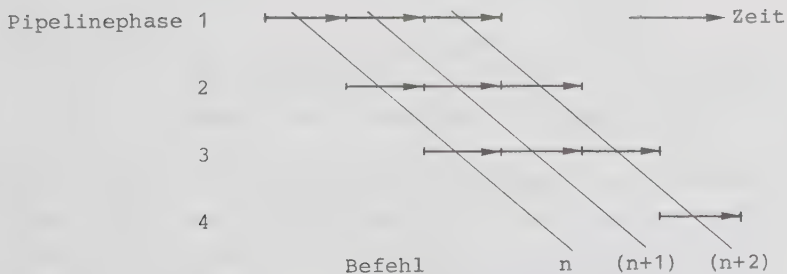
Bei diesem Befehlstyp wird ein NOP-Befehl automatisch von der Prozessorsteuerung in das Programm während der Programmbearbeitung eingefügt, ^{*)} so daß bei der Aufeinanderfolge von Befehlen dieses Typs der Kollisionsfall vermieden wird.

- (Register nach Register)-Typ :

Bei diesem Befehlstyp befinden sich die Operanden ausschließlich in den Registern der Zentraleinheit. Ein Beispiel wäre

ADD R1, R2, R2 .

Die Prozessorsteuerung des DSP erzwingt bei diesem Befehlstyp die vollständige Befehlsausführung in der dritten Pipelinephase. Es ergibt sich dadurch folgendes Schema:



Dabei ist der Befehl n und (n+1) vom beschriebenen Typ. Es kann somit in Phase drei des Befehls (n+2) zu keiner Kollision mit der Ausführung des Befehles (n+1) kommen.

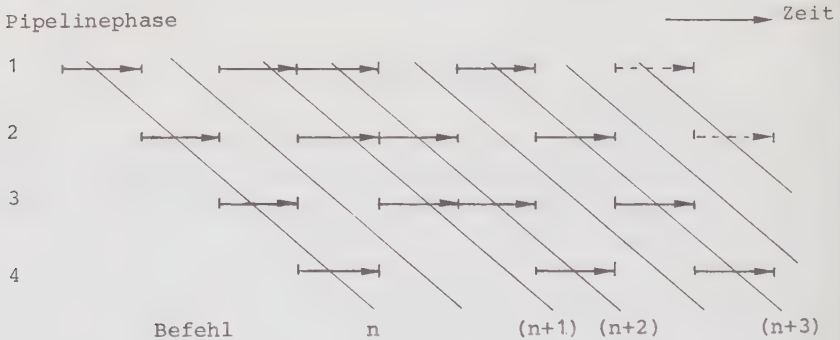
*) Dies kann vom Programmierer auch verhindert werden

Die Wirkung der beiden Verfahren soll zusammenfassend an einem Beispiel demonstriert werden. Es sei die Befehlsfolge

Befehl

n	ADD	<Speicher> , R0, R1
n+1	ADD	R1, R2, R2
n+2	ADD	R3, R4, <Speicher>
n+3	ADD	<Speicher>, R6, R7

zu bearbeiten. Dies führt zu folgendem Pipelinephasendiagramm:



Das Programm wird somit automatisch kollisionsfrei bearbeitet. Hierzu seien noch zwei Bemerkungen angefügt.

- Befehle, bei denen automatisch NOP-Instruktionen eingefügt werden, sind 2-Zyklus-Befehle. Vom Standpunkt der Befehlsdurchsatzrate werden sie nur halb so schnell wie die übrigen Befehle bearbeitet. Dies fällt jedoch kaum ins Gewicht, da die Rechenzeitkritischen Algorithmen für die Signalverarbeitung ohnehin meistens in der Arithmetischen Einheit realisiert werden.
- Die angegebene Steuerung berücksichtigt einen Kollisionsfall nicht: Wenn nach einem (Speicher zu Speicher)-Transportbefehl ein (Speicher nach Register)-Befehl folgt, z.B. in

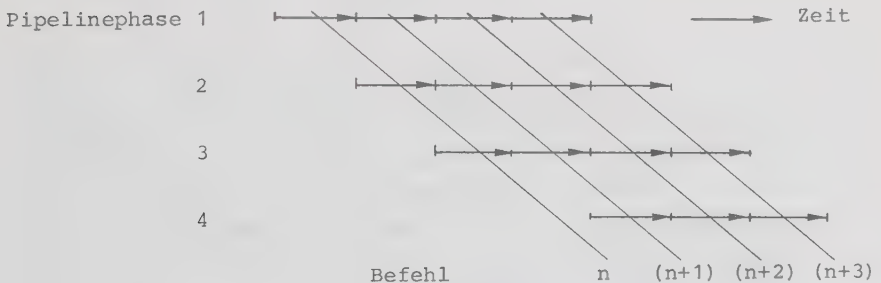
TRA Quelle 1, Ziel 1
 ADD Quelle 2, R1, R2 (5.3-2)
 mit (Quelle 1, Ziel 1, Quelle 2) = Speicheradressen

führt dies dann zum Kollisionsfall, wenn die Adresse Ziel 1 die selbe Buseinheit wie die Adresse Quelle 2 anspricht.

Bei der Programmerstellung gilt daher die Regel:
 Es müssen Befehlsfolgen wie (5.3-1) und (5.3-2) vermieden werden.

III. Programmverzweigungsfall

Die Programmverzweigung beim Auftreten bedingter Sprünge ist in einem Pipelinesystem kritisch, wie das folgende Phasendiagramm zeigt, in dem der Befehl (n+1) ein bedingter Sprungbefehl sei:



Dieses Diagramm ergibt sich zum Beispiel in der Befehlsfolge

n	SUB	R1, R2, R2 ;	<R1> - <R2> → R2
n+1	JEQ	MARKE ;	Springe nach MARKE, falls
			Ergebnis gleich Null war
n+2	ADD	R1, R2, R2 ;	<R1> + <R2> → R2 .

Bei der Programmerstellung bedeutet dies, daß in der Befehlsfolge

n	SUB	R10, R11, R11
n+1	SUB	R8, R10, R10
n+2	JEQ	MARKE 1
n+3	JEQ	MARKE 2

der Sprungbefehl (n+2) das Ergebnis der Subtraktion n ausgewertet, und der Befehl (n+3) das der Operation (n+1). In einem rein sequentiell arbeitenden Rechner hätte die angegebene Befehlsfolge keinen Sinn, da der Sprungbefehl (n+3) nie ausgeführt würde.

Soll ein bedingter Sprungbefehl das Verknüpfungsergebnis einer unmittelbar vorangegangenen Operation auswerten, so ist er von ihr durch einen NOP-Befehl zeitlich zu trennen:

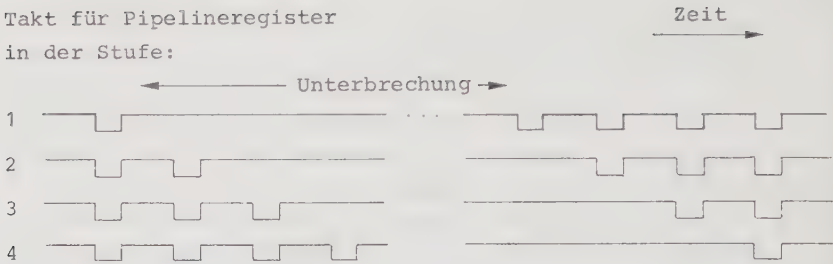
SUB	R1, R2, R2,
NOP	
JEQ	MARKE .

Diese, sowie die im letzten Abschnitt angegebene Regel beschreibt vollständig die Einschränkungen, die beim Programmieren des hier beschriebenen Systemes zu berücksichtigen sind.

IV. Unterbrechungsfall

Im Fall einer Programmunterbrechung (z.B. beim Auftreten eines Interruptes oder eines Programmstops bzw.-starts von der Bedienungskonsole aus) muß gewährleistet sein, daß Befehlsbearbeitungen vollständig abgeschlossen, bzw. mit der Phase 1 begonnen werden. Dadurch verbietet sich das einfache Aus- bzw. Einschalten des Rechnertaktes. Der Takt für die Register der Pipelinestufen (Bild (5.3-4)) muß nach folgendem Muster bereitgestellt werden:

Takt für Pipelineregister
in der Stufe:



c) Realisierung der wichtigsten Funktionsblöcke

Von den in Bild (5.3-4) angegebenen Funktionsblöcken werden nun die für die Leistungsfähigkeit der Zentraleinheit entscheidenden Untersysteme, nämlich die Programmsteuerung sowie der Adreßrechner, detailliert beschrieben.

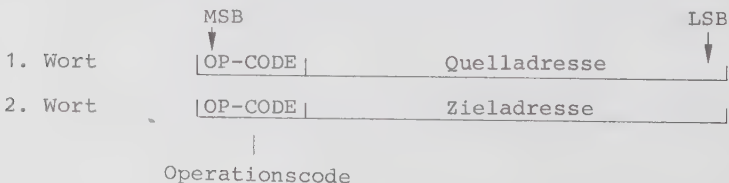
I. Programmsteuerung

Die Programmsteuereinheit bearbeitet die Befehlsholphase und ist damit für die Ausführung von Sprungbefehlen sowie Programmunterbrechungen zuständig.

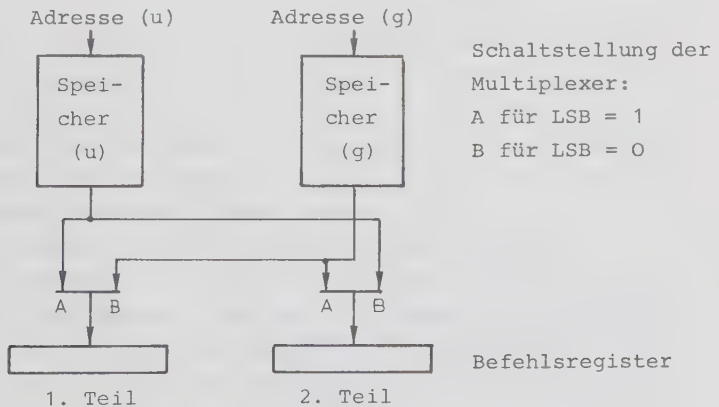
Die Befehlsholphase und die Bearbeitung von Sprungbefehlen soll in einem Maschinenzyklus durchführbar sein, dies wird gefordert von der Pipelinestruktur in Bild (5.3-5). Dadurch entsteht folgendes Problem: die Maschinenbefehle des DSP müssen z.T. mit zwei Befehlsworten codiert werden, um neben dem Operationscode auch Adressen an den Prozessor liefern zu können. Z.B. hat ein Datentransportbefehl für einen (Speicher nach Speicher)-Transfer folgende Form:

TRA <Quelladresse> , <Zieladresse>

und wird codiert durch :



Es müssen somit zwei Zugriffe auf den Programmspeicher in einem Maschinenzyklus von 150 nsec durchgeführt werden, vorher muß im gleichen Zyklus die Adresse des Befehls errechnet werden. Dies ist beim Aufbau des Systems mit Schottkyelementen nicht möglich. Im DSP wird deshalb der Programmspeicher aufgeteilt in zwei Speicherblöcke, von denen der eine durch die geraden, der andere durch die ungeraden Adressen angesprochen wird. Dadurch sind zwei Speicherzugriffe gleichzeitig möglich, allerdings wird die Adreßgenerierung erheblich aufwendiger, denn in einem Programm kommen Maschinenbefehle, die mit ein oder zwei Befehlsworten codiert sind, gemischt vor. Das bedeutet, daß sich das erste Befehlswort eines Doppelwortbefehls bzw. das Befehlswort eines Einfachwortbefehls in jedem der beiden Speichermoduln befinden kann. Deshalb muß ein Multiplexer in den Datenweg vom Programmspeicher zum Befehlsregister eingefügt werden, dessen Schaltstellung durch das Bit geringster Wertigkeit der Adresse gesteuert wird (g für gerade, u für ungerade):



Ia) Adreßrechnung

Im folgenden wird gezeigt, wie die Adressen für die beiden Speichermoduln errechnet werden. Dabei wird prinzipiell immer davon ausgegangen, daß der nächste Befehl ein Doppelwortbefehl ist, und es werden profilaktisch beide Worte ins

Befehlsregister transportiert. Dazu ist die Errechnung je einer Adresse für die beiden Speichermoduln erforderlich.

Der Programmzähler enthält die aktuelle Befehlsadresse. Die Berechnung der Folgeadresse ist abhängig vom aktuellen Befehl, die auszuführenden Adreßrechnungen zeigt folgende Tabelle (hierin bedeutet PC = Programmzähler, Offset = Adreßteil des Sprungbefehles, Interruptvektor = Interruptzieladresse):

aktueller Befehl	Folgebefehl	
	1. Wort	2. Wort
Einfachwortbefehl	$\langle PC \rangle + 1$	$\langle PC \rangle + 2$
Doppelwortbefehl	$\langle PC \rangle + 2$	$\langle PC \rangle + 3$
Skipbefehl (Einfachwort)	$\langle PC \rangle + 3$	$\langle PC \rangle + 4$
Skipbefehl (Doppelwort)	$\langle PC \rangle + 4$	$\langle PC \rangle + 5$
absoluter Sprungbefehl	Offset	Offset + 1
relativer Sprungbefehl	$\langle PC \rangle + \text{Offset}$	$\langle PC \rangle + \text{Offset} + 1$
(Interrupt)	Interruptvektor	Interruptvektor + 1

Bei der Bearbeitung bedingter Sprung- und Skipbefehle ist von der Programmsteuereinheit zunächst die Bedingung abzufragen, was einen Zeitaufwand von ca. 30 bis 40 nsec erfordert. Würde erst danach eine der in der Tabelle angegebenen Rechnungen zur Bestimmung der Adressen des nächsten Befehles durchgeführt werden, so ließe sich dies zusammen mit dem Speicherzugriff nicht in der geforderten Zeit von 150 nsec durchführen. In der Programmsteuereinheit des DSP werden deshalb profilaktisch die direkt folgenden Adressen, sowie die Skip- und Sprungadressen simultan errechnet, während eine Logik die Sprung- und Skipbedingungen abfragt.

Die Entscheidung, ob der aktuelle Befehl ein Einfachwort- oder Doppelwortbefehl, bzw. ein absoluter oder relativer Sprungbefehl ist, wird von einer einstufigen Gatterlogik des Dekodierers innerhalb von 5 nsec noch vor Errechnung der Folgeadressen getroffen. Dies wird möglich durch entsprechende Codierung der Befehle.

Die Struktur des Rechenwerkes in der Programmsteuereinheit zeigt Bild (5.3-6). Neben der Verwendung mehrerer Addierwerke ist die unkonventionelle Verwendung des Programmzählerregisters (PC) auffallend: im DSP enthält der Programmzähler am Ende eines Zyklus die Adresse des aktuellen Befehls, nicht die des folgenden. Dadurch kann eine Pipelinestufe eingespart werden, und die Programmsteuereinheit kann die gestellten Aufgaben innerhalb eines Zyklus erfüllen.

Die Multiplexer M1 bis M4 in Bild (5.3-6) werden in Abhängigkeit davon gesteuert, ob der aktuelle Befehl ein Einfach- oder Doppelwortbefehl ist. Die Bausteine M5 bis M8 realisieren die Umschaltung zwischen absoluten und relativen Sprungbefehlen, weiterhin gestatten sie die Übernahme eines Interruptvektors. Die Generierung dieser Adresse wird im nächsten Abschnitt beschrieben.

Durch die Abfrage der Sprungbedingungen, die das Statusregister liefert, werden die Datenwege durch die Multiplexer M9 und M10 ausgewählt. M11 wird durch das Bit geringster Wertigkeit der vom Multiplexer M9 gelieferten Adresse gesteuert: für $LSB = 0$ läßt er die Adresse von M9 zum Speicher (g) passieren, für $LSB = 1$ die von M10. Dieses Bit gelangt nicht zu den Speichermodulen, ihr Adreßwort ist um ein Bit verkürzt gegenüber der im Programm-Zählerregister enthaltenen und für die Addierwerke bereitgestellten Befehlsadresse.

Zur Realisierung von Befehlen, die Sprünge in, bzw. Rücksprünge aus einem Unterprogramm erlauben, ist das Programmzählerregister PC mit einem Stackspeicher versehen, in dem Rücksprungadressen abgelegt werden können.

Es sei noch bemerkt, daß die Kapazität der beiden Programmspeichermodulen jeweils 2048 Befehlsworte beträgt.

Bild (5.3-6) Struktur des Rechenwerkes für die Befehlsfolgeadresse

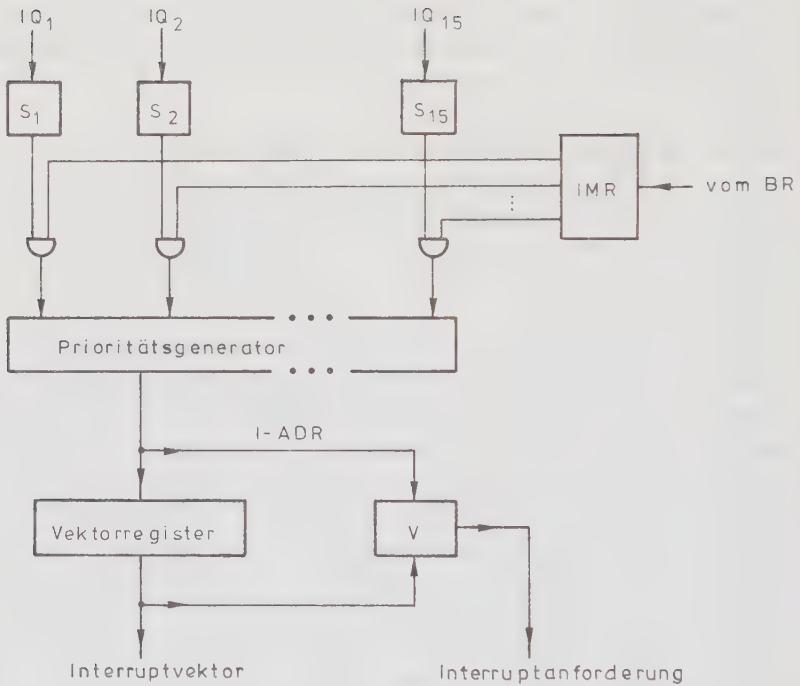
Ib) Interruptsteuerung

Die im DSP implementierte Programmsteuereinheit besitzt eine vergleichsweise aufwendige Interruptsteuerung.

Ein außerhalb des Prozessors auftretender Impuls (= Interruptanforderung) bewirkt, daß das Programm an einer bestimmten Programmstelle fortgesetzt wird, daß also der natürliche Programmlauf unterbrochen wird.

Da in der vorliegenden Abhandlung jedoch in erster Linie Datenverknüpfungsstrukturen und keine Steuerungen behandelt werden, wird die Interruptsteuerung nur kurz gestreift. Ihre wesentlichen Funktionsblöcke sind in Bild (5.3-7) zu sehen.

Auftretende Interruptanforderungsimpulse der Interruptquellen (z.B. Fernschreiber, ext. Abtasttakt) werden in Speicherflipflops festgehalten. Entsprechend dem Inhalt des Interruptmaskenregisters werden bestimmte Interruptquellen ausgeschlossen. Der Prioritätsgenerator liefert die Adresse (4 Bit) der Interruptquelle, die die aktuell höchste Priorität besitzt. Diese Adresse wird in einem Vergleicher mit der im Vektorregister gespeicherten Adresse verglichen. Wenn sie höher ist, erzeugt der Vergleicher eine Interruptanforderung für die Programmsteuereinheit. Dabei wird die neue Adresse aus dem Prioritätsdekoder in das Vektorregister geladen. Auf die Darstellung weiterer Einzelheiten sowie der umfangreichen Steuerungsvorgänge wird hier verzichtet.



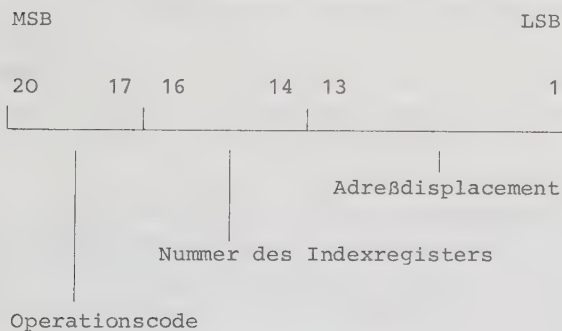
- IQ_v = Interruptquelle v
 S_v = Interruptspeicherflipflop
IMR = Interruptmaskenregister
I-ADR = Adresse der aktuell mit höchster Priorität
anliegenden Interruptanforderung
V = Vergleicher

Bild (5.3-7) Interruptsteuerung

II. Adreßrechner

Die Aufgabe des Adreßrechners besteht in der Bereitstellung von Adressen für den Quell- und Zieladreibus (Bild (5.3-1)).

Für die Beschreibung der Struktur des Adreßrechners wird ausgegangen von dem Codierungsformat des Adreßteils eines Maschinenbefehls. Es hat folgende Form:



Der Adreßrechner errechnet aus dem im Befehl mitgeteilten Adreßdisplacement unter Zuhilfenahme des angegebenen Indexregisters die physikalische Adresse gemäß dem im Operationscode gewünschten Adreßmodus.

Zunächst werden die möglichen Anweisungen an den Adreßrechner beschrieben, danach wird seine Struktur angegeben.

IIa) Mögliche Adreßmodifikationen

Es stehen im Adreßrechner jeweils 7 Index-, Modulo- und Offsetregister zur Verfügung. Wird im Adreßteil für die Nummer des Indexregisters Null angegeben, so wird das Adreßdisplacement direkt als physikalische Adresse interpretiert. In allen anderen Fällen entscheidet der Operationscode über die Errechnung der physikalischen Adresse. Dabei gilt immer:

Physikalische Adresse = <Indexregister> + Adreßdisplacement

Neben dieser Adreßrechnung führt der Adreßrechner noch folgende Operationen alternativ aus:

1. Inkrementieren des Indexregisters
2. Dekrementieren des Indexregisters
3. Inkrementieren des Indexregisters und Skip,
falls <Indexregister> = 0
4. Dekrementieren des Indexregisters und Skip,
falls <Indexregister> = 0
5. Inkrementieren des Indexregisters und Skip,
falls <Indexregister> = <Offsetregister>
6. Dekrementieren des Indexregisters und Skip,
falls <Indexregister> = <Offsetregister>.

Wenn der Adreßteil eines Maschinenbefehls eine Quelladresse bestimmt, kann die vom Adreßrechner errechnete Adresse umgeformt werden in Bit-Reversal-Form nach Abschnitt 4.1.3.

Das angegebene Inkrementieren bzw. Dekrementieren eines Indexregisters wird dabei immer in Moduloarithmetik bezüglich dem Inhalt eines zugeordneten Moduloregisters gerechnet, so daß die in Abschnitt 4.2.1 geforderten Adreßalgorithmen realisiert werden.

Weiterhin gestattet der Adreßrechner auch die Addition einer Konstanten zu einem Indexregister, wobei auch diese Addition in Moduloarithmetik errechnet wird. Durch Löschen des einem Indexregister zugeordneten Moduloregisters kann die Rechnung in Moduloarithmetik unterbunden werden.

Zur Verdeutlichung des Gesagten wird nun noch ein konkretes Beispiel eines Maschinenbefehls angegeben, bei dem die Möglichkeiten des Adreßrechners weitgehend genutzt werden. Als Beispiel wird der Befehl

ADD 1000 (I1)ISZ↑, RO, R1

in Assemblerschreibweise angegeben. Er bewirkt die Addition einer Speichervariablen mit dem Inhalt des Registers 0, das Ergebnis wird in Register 1 abgelegt.

Diese Operation wird vom Mikroprozessor in Bild (5.3-4) durchgeführt. Der Adreßrechner errechnet die physikalische Quelladresse durch Addition der Zahl 1000 (= Adreßdisplacement) zum Indexregister 1 (I1). Der Pfeil bedeutet eine Umrechnung dieser Adresse in bitreversed Form. Der Ausdruck "ISZ" stellt den Operationscode für den Adreßrechner dar. Er bedeutet: das Indexregister 1 soll inkrementiert werden in Moduloarithmetik bzgl. des Inhalts des Moduloregisters 1, weiterhin soll der nächste Befehl übersprungen werden (Skip) falls das Indexregister 1 vor der Inkrementierung gleich Null war. Der gesamte Maschinenbefehl wird in zwei Zyklen, also in 300 nsec bearbeitet.

IIb) Adreßrechnerstruktur

Bild (5.3-8) zeigt die Struktur des Adreßrechners. Der Übersichtlichkeit wegen sind lediglich die Elemente zur Berechnung einer Adresse eingezeichnet. Für Datentransportbefehle vom (Speicher-nach-Speicher)-Typ muß eine Quell- und Zieladresse errechnet werden. Dies geht aus dem Bild nicht hervor. Weiterhin sind die Datenpfade zum Beschreiben der Register mit Daten aus dem Befehlsregister nicht eingezeichnet.

Jedes der angegebenen Registerfelder beinhaltet 7 Register, sie werden durch die Indexregisternummer im Adreßteil eines Maschinenbefehls adressiert.

Der Multiplexer M1 erlaubt die Realisierung der Basisoperation für das gewählte Indexregister, indem entweder die Zahlen 0, +1, -1 oder ein vom Befehlsregister geliefertes Zahlenwort durch das Addierwerk A1 zum Inhalt des Indexregisters addiert wird.

Die Moduloarithmetik wird durch das Addierwerk A2 sowie den Multiplexer M2 verwirklicht. Es wird prinzipiell vom Ausgangswert X des Addierers A1 der Inhalt des Moduloregisters M subtrahiert. Ist das Subtraktionsergebnis $X - M$ größer Null, so wird das Indexregister über den Multiplexer M2 damit geladen, andernfalls gelangt das Signal X zum Indexregister.

Diese Realisierung ist keine "echte" Moduloarithmetik.
Dazu wäre notwendig, nach der Vorschrift

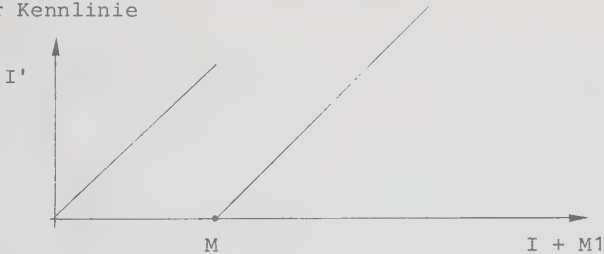
$$I \bmod M = I - n M \quad (5.3-3)$$

mit I = Inhalt des Indexregisters, M = Inhalt des
Moduloregisters, $n \in \mathbb{N}_0$ und n so, daß
 $0 \leq I - nM < M$ gilt

zu verfahren. Die angegebene Schaltung realisiert jedoch
nur folgende Zuordnung:

$$I' = \begin{cases} I + M1 & \text{für } (I+M1-M) < 0 \\ I + M1 - M & \text{für } (I+M1-M) \geq 0 \end{cases} \quad (5.3-4)$$

mit $M1$ = Ausgangssignal des Multiplexers $M1$. Dies führt
zu der Kennlinie



die die Moduloarithmetik nur für $n = 0$ und $n = 1$ richtig
durchführt, was gleichbedeutend ist mit der Notwendigkeit

$$M1 \stackrel{!}{\leq} M$$

Dies ist beim Inkrementieren und Dekrementieren immer erfüllt.
Ist jedoch $M1 = M + \Delta$ mit $\Delta > 0$, also $M1 > M$,

so liefert die Zuordnung (5.3-4)

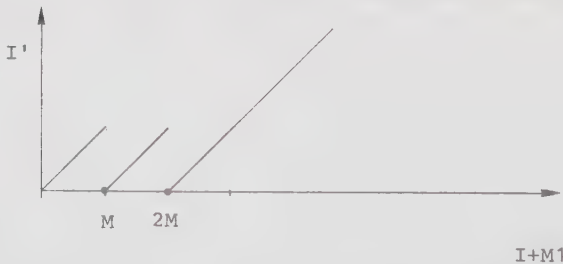
$$I' = I + \Delta$$

weil $I \geq 0$ ist und damit auch $I + \Delta > 0$ ist. Dieses Ergebnis ist nach Gleichung (5.3-3) falsch, wenn $I + \Delta \geq M$.

Für den Zahlenbereich $0 < \Delta < M$ für Δ ist eine richtige Lösung dann zu erhalten, wenn die in Gleichung (5.3-4) angegebene Operation zunächst mit $M1 = M + \Delta$, dann mit $M1 = 0$ durchgeführt wird. Bei der Programmerstellung muß also zweimal der Additionsbefehl aufgerufen werden:

```
ADD    M + Δ, I ; Addition von M+Δ zum Inhalt des
                Indexregisters
ADD     0, I ; Korrekturaddition ,
```

und man erhält hierfür die unten angegebene Kennlinie.



Jedem Indexregister ist neben dem Modulo- und Offsetregister noch ein Statusregister zugeordnet. Es beinhaltet die Ergebnisse der Vergleiche des Indexregisterinhaltes mit der Zahl Null und dem Inhalt des Offsetregisters. Dadurch können an die Programmsteuereinheit Aufforderungen zu Skipbefehlen weitergegeben werden.

Das Addierwerk A3 realisiert die Addition des Adreßdisplacements zum Indexregisterinhalt. Das Additionsergebnis kann nachfolgend durch eine Logik für Bitreversal modifiziert werden.

Entscheidend für die Programmerstellung ist die Eigenschaft des Adreßrechners, daß die Modifikationen der Indexregister für den aktuellen Befehl nicht wirksam werden. Wenn dies erreicht werden soll, müßte das Addierwerk A3 nicht den Inhalt des Indexregisters zum Adreßdisplacement addieren, sondern den Ausgangswert des Multiplexers M2. Dann aber würden sich die Additionszeiten der Addierwerke A1, A2 und A3 aufsummieren, so daß die Bereitstellung der physikalischen Adresse nicht mehr in einem Maschinenzyklus möglich wäre und sich der Adreßrechner nicht in der Pipelinestruktur von Bild (5.3-5) einsetzen ließe.

Es sei hier nur am Rande erwähnt, daß die Signalschleife Indexregister-Addierwerk A1 - Addierwerk A2 - Multiplexer M2 - Indexregister neben der Realisierung bedingter Sprungbefehle in der Programmsteuereinheit der zeitkritischste Signalweg im DSP ist.

5.3.1.3 Speichermoduln

Die in Bild (5.3-1) angegebenen Speichermoduln zeichnen sich aus durch ihre Adressierbarkeit von zwei Adreßquellen: dem Quelladreßbus und dem Zieladreßbus. Ihre Struktur zeigt Bild (5.3-9).

Der Multiplexer M läßt entweder die Quell- oder die Zieladresse zum Speicher passieren, er wird gesteuert von den beiden Logik-elementen zur Adreßerkennung. Die letztgenannten initialisieren zudem die Speicherschreib- bzw.-lesezyklen.

Es sei lediglich erwähnt, daß dem System ein weiterer externer Hauptspeicher mit einer Kapazität von 256 kByte zur Verfügung steht. Dieser externe Speicher ist ein 2-Torspeicher, dessen Inhalt ständig zyklisch ausgelesen und auf einem Bildwiedergabegerät als Halbtonbild dargestellt wird. Dies ermöglicht die Realisierung der in Abschnitt 3.2.5 und 3.2.6 dargestellten Algorithmen zur Bildverarbeitung.

1. einen Lochstreifenleser mit einer Steuereinheit für den direkten Speicherzugriff (DMA) auf den Daten- und Programmspeicher,
2. ein Floppy-Disk-Laufwerk,
3. eine Parallelschnittstelle zum Datenaustausch nach dem Hand-shake-Prinzip,
4. einen Lochstreifenstanzer,
5. die Kopplung mit einem externen Rechner,
6. schnelle serielle Schnittstellen für den Datenaustausch mit externen Signalverarbeitungssystemen, und für
7. einen Bedienungsfernsehreiber, wobei für dessen Steuerung ein separater Textspeicher zur Verfügung steht.

Das Prinzip beim Entwurf der Ein-/Ausgabeeinheit lag in der weitgehenden Automatisierung der Interfacesteuerung, um deren Handhabung zu erleichtern und Programmspeicherkapazität einzusparen.

5.3.1.5 Konsolensteuerung

Beim Entwurf und dem Aufbau des DSP wurde eine vollständige Steuerbarkeit von einer Bedienungskonsole angestrebt. Dies bedeutet, daß sämtliche Dateninhalte der Register der Zentraleinheit sowie der am Systembus angeschlossenen Einheiten angezeigt und modifiziert werden können.

Beim Aufbau eines einzelnen Systems bietet dies eine entscheidende Hilfe beim Test der verschiedenen Schaltungsteile, weiterhin ist eine Bedienungskonsole nützlich für die Fehlersuche beim Programmmentwurf.

Für die Integration einer universellen Konsolensteuerung in die Schaltung eines Rechners ist ein vergleichsweise hoher Aufwand notwendig, da sämtliche internen Datenwege und Steuersignale von außen beeinflußbar sein müssen.

5.3.1.6 Technische Daten

Die Beschreibung der Systemarchitektur des Digitalen Signalprozessors wird abgeschlossen durch eine tabellarische Zusammenfassung seiner technischen Daten.

a) Arithmetische Einheit

Datenwortbreite	: 20 Bit
Taktzeit	: 70 nsec
Rechenzeit für einen Zyklus	: 1950 nsec
Koeffizientenspeicher	: 2-Tor-Speicher für 2K Datenworte
Schaltungsstruktur	: 2 separate Rechenwerke zur Produktsummenbildung

$$y_{1,2} = \sum_{p=1}^3 a_p x_p, \text{ aufgebaut}$$

in "Verteilter Arithmetik",

Addierwerk für $y_3 = y_1 + y_2$

Technologie	: Schottky-TTL-Bausteine (SSI, MSI)
-------------	-------------------------------------

b) Zentraleinheit

Datenwortbreite, Befehlswortbreite: 20 Bit

Zykluszeit : 150 nsec

Rechenzeit für:

(Register nach Register)-Transfer : 150 nsec

(Speicher nach Speicher)-Transfer : 150 nsec

(Register nach Register)-Addition : 150 nsec

(Register/Speicher)-Addition : 300 nsec

Befehlsstruktur : 3-Adreß-Befehle

Steuerstruktur : Pipelinerechner mit 4 Pipeline-
stufen, direkte Hardware-
steuerung,
vollsynchrone Bussteuerung

Synchronisationsmechanismen: Taktsynchronisation,
Interruptsynchronisation
Interruptsteuerung : 15 maskierbare Interrupt-
quellen mit Prioritätszuweisung
Schaltungsstruktur : autonome Untersysteme zur Pro-
grammsteuerung, Adreßberechnung,
ALU-Funktionsausführung
Technologie : Schottky-TTL-Bausteine (SSI, MSI, LSI)

c) Speicher

I. Datenspeicher : 4 Speichermoduln, 2 für 2K,
2 für 4K Datenworte
II. Programmspeicher : 2 Speicherblöcke für jeweils
2K Befehlsworte
Zykluszeit : 150 nsec
Technologie : Schottky-TTL-Bausteine (LSI)

d) Ein-/Ausgabeeinheit

I. Langsame, serielle Schnittstellen:

4 serielle Schnittstellen für
20 mA Schleifenstrom, für Baudraten
bis 200 kBaud. Eine Schnittstelle
mit Textspeicher für 4K ASCII-Zeichen.

II. Schnelle, serielle Schnittstellen:

jeweils 6 Ein- und Ausgabeschnitt-
stellen für 16-Bit-Worte bei Daten-
raten bis zu 320 kHz

III. Parallele Schnittstelle: Handshakeschnittstelle,

Prozedurdauer: 1 μ sec

IV. Lochstreifenleserinterface:

mit Steuerung für 1. DMA-Betrieb
zum automatischen Programmeinlesen,
2. automatischer Programmsuchlauf.

e) Assembler

Auf der Basis eines Assemblerprogrammes für einen Prozeßrechner wurde durch Umdefinition der Operationscodetabellen und Adreßmodi ein Crossassembler entwickelt [42].

5.3.2 Anwendungsbeispiele

5.3.2.1 Eindimensionale Signalverarbeitung

Für den Digitalen Signalprozessor wurden erste Programme zur Signalverarbeitung entwickelt, deren wichtigste technische Daten in folgender Tabelle angegeben werden.

	nichtrek. Filter	rekursives Filter	FFT
Filtergrad	$n = 6 \dots *)$	$n = 2 \dots *)$	-
Feldlänge	-	-	$M=2^m, m=1\dots 9$
Programmlänge	11 Befehle für $n=6$	10 Befehle pro Block 2.Grades	52 Befehle für Butterfly
Auslastung der Arithm. Einheit	$n = 6 : 46\%$ $n = 30 : 81\%$	$n = 2 : 54\%$ $n = 16 : 95\%$	89% $\forall M$
Rechenzeit	$(\frac{n}{6} 13+15) \cdot$ 0,15 μsec	$(\frac{n}{2} 13+11) \cdot$ 0,15 μsec	$(30 \cdot \frac{M}{2} \cdot 1dM) \cdot$ 0,15 μsec

Das Programm zur Berechnung eines nichtrekursiven Filters nutzt die Rechenleistung der Arithmetischen Einheit insbesondere dann optimal aus, wenn für den Filtergrad gilt:

$$n = k \cdot 6, \quad k \text{ ganzzahlig}.$$

Zur Berechnung eines rekursiven Filters wird die Struktur nach Bild (3.2-6) realisiert.

Bei der Berechnung der schnellen Fourier-Transformation werden nicht die höchsten möglichen Geschwindigkeiten erreicht, da das Programm bezüglich der Feldlänge variabel ist und deshalb eine vergleichsweise umfangreiche Schleifenverwaltung nötig ist.

*)

Der Filtergrad ist praktisch nicht begrenzt, da er nur von der Kapazität des Daten- und Koeffizientenspeichers abhängt.

Ein Butterfly wird während zwei Rechenzyklen der Arithmetischen Einheit berechnet [41].

Der Vergleich der angegebenen Rechenzeiten mit denen des in Abschnitt 5.2 angegebenen Systems zeigt, daß die Programmlaufzeiten des Digitalen Signalprozessors in der gleichen Größenordnung liegen, obwohl er eine mittlere Multiplikations-Akkumulationszeit von 325 nsec (gegenüber 200 nsec beim vorgenannten System) hat, und zusätzlich die Möglichkeit zur Programmierung auf der wesentlich einfacheren Maschinenbefehlsebene gibt.

Ein Rechnervergleich, bei dem lediglich Programmlaufzeiten berücksichtigt werden, ist natürlich unvollständig, da z.B. Möglichkeiten für umfangreiche Programmsysteme wie Betriebssysteme oder Fehlersuchprogramme, oder auch die Möglichkeiten der Ein- Ausgabe unberücksichtigt bleiben.

Es ist z.B. in dem in Abschnitt 5.2 beschriebenen Minimal-system die Berechnung der schnellen Fouriertransformation nur sehr eingeschränkt möglich, da es über keine Einheiten für die Adreßberechnung verfügt.

5.3.2.2 Mehrdimensionale Signalverarbeitung

Zur Berechnung der in Abschnitt 3.2-5 und -6 beschriebenen Verfahren zur mehrdimensionalen Signalverarbeitung wurde ein Programmsystem entwickelt, das den Ausdruck

$$\mathbf{Y}_3 = \underline{\mathbf{G}} \mathbf{Y}_2 + \mathbf{Y}_1 \quad (5.3-5)$$

sowie

$$\mathbf{Y}_3 = \underline{\mathbf{G}} \mathbf{Y}_2 \quad (5.3-6)$$

berechnet. Dazu wurde der Algorithmus aus Abschnitt 3.2.6 verwendet. Gleichung (5.3-5) realisiert dabei direkt die in Bild (3.2-28) angegebene Struktur. Die in den Gleichungen (5.3-5) und (-6) angegebenen Operationen lassen sich vorteilhaft auch zur Berechnung von Systemen mit höherem Blockgrad einsetzen. Aus [24] ist folgende Struktur entnommen.

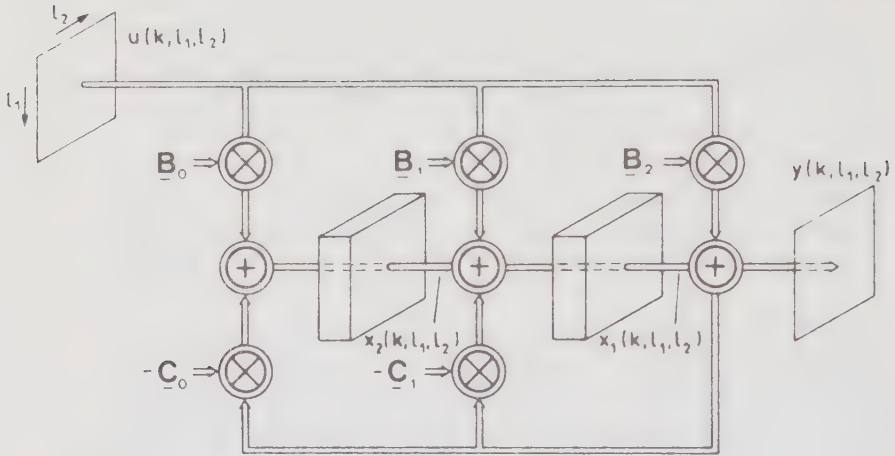


Bild (5.3-10) System zur rekursiven Bildfolgenverarbeitung vom Blockgrad 2

Sie wird realisiert durch:

SCHLEIFE: $\mathbf{y}(k) := \mathbf{B}_2 \mathbf{u}(k) + \mathbf{x}_1(k)$
 $\mathbf{x}_1(k+1) := \mathbf{B}_1 \mathbf{u}(k) + \mathbf{x}_2(k)$
 $\mathbf{x}_1(k+1) := -\mathbf{C}_1 \mathbf{y}(k) + \mathbf{x}_1(k+1)$
 $\mathbf{x}_2(k+1) := \mathbf{B}_0 \mathbf{u}(k)$
 $\mathbf{x}_2(k+1) := -\mathbf{C}_0 \mathbf{y}(k) + \mathbf{x}_2(k+1)$
 Sprung nach SCHLEIFE

Die Datenfelder $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{u}$ und $\mathbf{y}$ werden bei der hier beschriebenen Realisierung des Algorithmus mit dem Digitalen Signalprozessor in einem externen Speicher gehalten. Die nachfolgend angegebenen Rechenzeiten beziehen sich auf eine Feldgröße von jeweils 256 x 256 Bildpunkte, und ein Koeffizientenfeld der Größe 3 x 3:

Errechnung des Ausdruckes (5.3-5)

Rechenzeit $T_R = 345 \text{ msec}$

Datentransportzeit in externen Speicher

$T_{TR} = 870 \text{ msec}$

Errechnung des Ausdruckes (5.3-6)

Rechenzeit $T_R = 345 \text{ msec}$

Datentransportzeit in externen Speicher

$T_{TR} = 580 \text{ msec} \quad .$

6. Zusammenfassung

In der vorliegenden Arbeit werden die Zusammenhänge zwischen repräsentativen Algorithmen der digitalen Signalverarbeitung und den Strukturen spezieller Rechenwerke untersucht.

Nach der Definition der Anwendungsklassen digitaler Systeme wird eine Auswahl häufig interessierender Algorithmen auf ihre Zerlegbarkeit in simultan berechenbare Aufgaben analysiert. Für diese Teilaufgaben werden realisierende Strukturen von Teilprozessoren ermittelt, die entweder durch ein Programm in einem Rechenwerk, oder durch die Kaskadierung mehrerer Rechenwerke zu verwirklichen sind.

Dabei wird versucht, die hier behandelten Algorithmen im Kern auf die Berechnung von Produktsummen zurückzuführen. Dies gestattet dann die Angabe bezüglich der Algorithmen universeller Hardwarestrukturen der Teilprozessoren.

Neben den so gewonnenen spezifischen Struktureigenschaften werden allgemeinere Anforderungen an Signalprozessoren formuliert, die insbesondere die Adreßgenerierung sowie die Synchronisationsmechanismen betreffen.

Es wird dann ein zelluläres Multiprozessorsystem vorgeschlagen, das die hardwaremäßige Kaskadierung mehrerer Teilprozessoren realisiert. Für ein bereits aufgebautes Minimalsystem der angegebenen Teilprozessorstruktur werden erreichte Rechenleistungen und technische Daten angegeben.

Sodann wird ein universeller digitaler Signalprozessor beschrieben, dessen Arithmetische Einheit in ihrer Struktur bis zu 6 Teilprozessoren kompakt realisiert. Es wird gezeigt, wie dieser Baustein durch ein schnelles, angepaßtes Universalrechenwerk mit Daten ver- und entsorgt werden kann. Es arbeitet in einer vierstufigen Pipelinestruktur. Detaillierte Beschreibungen der Untersysteme zur Programmsteuerung sowie zur Adreß-

generierung werden angegeben. Dabei zeigt sich, daß durch eine angemessene Struktur dieser Einheiten die durch das Pipelineverfahren entstehenden Probleme stark zurückgedrängt werden können, so daß der Benutzer nur wenige Regeln bei der Programmerstellung berücksichtigen muß.

Die beschriebene Architektur des Digitalen Signalprozessors gestattet die Programmierung auf Maschinenbefehls-ebene und gewährleistet trotzdem vergleichsweise hohe Rechengeschwindigkeiten.

Es werden abschließend die technischen Daten des aufgebauten Systems, sowie einige Beispiele für erreichte Rechenzeiten angegeben.

7. Literaturverzeichnis

- [1] H.W. Schüßler:
Digitale Systeme zur Signalverarbeitung, Springer-Verlag, Berlin, Heidelberg, New York, 1973.
- [2] CSPI, Hersteller:
An Introduction to the MAP Series, Massachusetts, 1975.
- [3] Floating Point Systems, Hersteller:
AP 120, Array Transform Processor, Oregon, 1975.
- [4] PCS, Hersteller:
Array Processor System APS, München, 1977.
- [5] CDA, Hersteller:
Micro Signal Processor MSP3, Massachusetts, 1978
- [6] Nicolet, Hersteller:
Zweikanal FFT-Analysator 660, Northrale, 1977.
- [7] Spectral Dynamics Corporation, Hersteller:
Digital Signal Processor SD 360, San Diego, 1976.
- [8] Gold, et.al.:
The FDP, A Fast Programmable Signal Processor, IEEE Trans. on Computers, C-20, 1971.
- [9] Mori, R., et.al.:
A Special-Purpose Computer for Digital Signal Processing, IEEE Trans. on Computers, C-24, 1975.
- [10] Rogers, J.A.V.:
GASP A Programmable Signal Processor, ASSP Conference, 1976.
- [11] Gold, B.:
The Design and Construction of a High Speed Sequential General Purpose Digital Processor and its Application to a Variety of Speech Processing Algorithms, MIT, Lincoln Laboratory, 1975.

- [12] Büttner, M.:
Interface für externe Prozessoren zur PDP 11/45,
Interne Mitteilung am Lehrstuhl für Nachrichtentechnik, Erlangen, 1975.

- [13] Seitzer, D.:
Elektronische Analog-Digital-Umsetzer,
Springer-Verlag, Berlin, Heidelberg, New York, 1977.

- [14] Oppenheim, V.; Schafer, R.:
Digital Signal Processing, Prentice-Hall Inc.,
New Jersey, 1975.

- [15] Schramm, H.:
Aufbau eines 16-Bit Mikrocomputers und seine Programmierung für die digitale Signalverarbeitung,
Diplomarbeit am Lehrstuhl für Nachrichtentechnik, Universität Erlangen-Nürnberg, 1979.

- [16] Dehner, G.:
Ein Beitrag zum rechnergestützten Entwurf rekursiver digitaler Filter minimalen Aufwands, Ausgewählte Arbeiten über Nachrichtensysteme Nr. 23, Universität Erlangen-Nürnberg, 1976.

- [17] Büttner, M.:
Untersuchungen über Grenzyklen in digitalen Filtern, Ausgewählte Arbeiten über Nachrichtensysteme Nr. 26, Universität Erlangen-Nürnberg, 1977.

- [18] Schüßler, H.W.:
Persönliche Mitteilung

- [19] Eckhardt, B.; Winkelkemper, W.:
Ein Beitrag zum Schaltungsentwurf digitaler Filter für den Tonfrequenzbereich. NTZ, Heft 1, 1974.

- [20] Rabiner, L.; Gold, B.:
Theory and Application of Digital Signal Processing,
Prentice-Hall Inc., New Jersey, 1975.
- [21] Schüßler, H.W.:
Digitale Signalverarbeitung, Skriptum des Lehrstuhls
für Nachrichtentechnik, Universität Erlangen-Nürnberg,
1976.
- [22] Schüßler, H.W.:
Persönliche Mitteilung
- [23] Abramowitz, M.; Stegun, A.:
Handbook of Mathematical Functions, Dover Publications
Inc., New York.
- [24] Schüßler, H.W.:
A 1D Approach to 2D Signal Processing, Erlangen, 1979,
erscheint demnächst.
- [25] Barber, D.L.A.:
"Digital Interfacing" in Minicomputer Interfacing,
Miniconsult, London, 1973.
- [26] Kober, R.; Kopp, H.; Kuznia, Ch.:
SMS 101-A Structured Multimicroprocessor System with
Deadlock-Free Operation Scheme, Euromicro Newsletter,
April 1976, Vol. 2, Nr. 2.
- [27] Salazar, A.C.; Hrsg.:
Digital Signal Computers & Processors, John Wiley & Sons
Inc., New York, 1977.
- [28] Allen, J.:
Computer Architecture for Signal Processing,
Proceedings of the IEEE, vol. 63, No. 4, 1975.
- [29] Klar, R.:
Digitale Rechenautomaten, Göschen, Band 1241/1241a,
Berlin, 1970.

- [30] Schloß, J.; Ballenberger, G.:
Ein modulares System zur Signalverarbeitung.
In Vorbereitung.

- [31] Ballenberger, G.:
Aufbau und Anwendung eines Akkumulator-Multiplizierers für die digitale Signalverarbeitung,
Studienarbeit am Lehrstuhl für Nachrichtentechnik,
Universität Erlangen-Nürnberg, 1979.

- [32] Klar, R.; Wichmann, H.:
Mikroprogrammierung, Arbeitsbericht des IMMD, Band 8,
Nr. 3, Universität Erlangen-Nürnberg, 1975.

- [33] Schloß, J.; Müller, H.:
Ein Entwicklungssystem für bipolare Mikroprozessorelemente, Elektronik, Heft 8, 1978.

- [34] Kolb, H.J.; Schloß, J.:
Ein Mikrosignalprozessor in BIT-SLICE-Technik mit externem Multiplizierer, Ausgewählte Arbeiten über Nachrichtensysteme, Nr. 38, Erlangen, 1979.

- [35] TRW, Hersteller:
Datenblatt "Multiplier-Accumulator TDC 1010J",
California, 1978.

- [36] Advanced Micro Devices, Hersteller:
Datenblatt "Bipolar Microprocessor Am 2903", 1978.

- [37] Schüßler, H.W.:
Persönliche Mitteilung

- [38] Büttner, M.:
Die Arithmetische Einheit des Digitalen Signalprozessors,
Interne Mitteilung am Lehrstuhl für Nachrichtentechnik,
Universität Erlangen-Nürnberg, 1978.

- [39] Büttner, M.; Schüßler, H.W.:
On Structures for the Implementation of the
Distributed Arithmetic, NTZ, Bd. 29, 1976.

- [40] Schloß, J.:
Aufbau und Funktion eines Digitalen Signalprozessors,
Ausgewählte Arbeiten über Nachrichtensysteme.
In Vorbereitung.

- [41] Czarnach, R.:
Ein Programmsystem zur Berechnung der ein- und zwei-
dimensionalen schnellen Fouriertransformation auf dem
DSP, Interne Mitteilung am Lehrstuhl für Nachrichten-
technik, Erlangen, 1979.

- [42] Dehner, G.F.:
Ein Crossassembler für den digitalen Signalprozessor.
Interne Mitteilung am Lehrstuhl für Nachrichtentechnik,
Erlangen, 1979.

L E B E N S L A U F

5. Juni 1950 geboren in Schwäbisch Hall
Vater: Franz-Ernst Schloß, Berufsberater
Mutter: Dorothea Irene, Fotografin
- 1957 - 1958 Grundschule in Reutlingen
- 1958 - 1960 Grundschule in Karlsruhe
- 1960 - 1964 Sichelgymnasium in Balingen
- 1964 - 1967 Goethe-Gymnasium in Karlsruhe
- 1970 Abitur am Martin-Behaim-Gymnasium in
Nürnberg
- 1970 - 1975 Studium der Elektrotechnik an der Universität
Erlangen-Nürnberg, Schwerpunkt Nachrichten-
technik/Elektronik.
Fachpraktikum bei den Firmen Grundig/Fürth
und Emac/Nürnberg
- 1975 Abschluß des Studiums mit einer Diplomarbeit
aus dem Gebiet der Datenübertragung
- 1975 - 1978 Wissenschaftlicher Mitarbeiter am Lehrstuhl
für Nachrichtentechnik der Universität
Erlangen-Nürnberg
- seit 1978 Wissenschaftlicher Assistent an diesem
Lehrstuhl

Copy-Center 2000

852 ERLANGEN · Obere Karlstr. 30
Tel. 09131-26197